

Notes on asy

F.A.Zhang(cvgmt)



版权所有 ©F.A.Zhang (cvgmt@163.com)

在 GNU General Public License (GNU 通用公共许可证) 的条款下授予复制、发布和/或修

改此文档的许可, 见 <http://code.google.com/p/asy4cn> 源代码包里面的 LICENSE 文件

本文档的下载地址 <http://code.google.com/p/asy4cn/downloads/list> 2010 年 12 月 25 日

编写说明

本文档讨论了功能强大的科技绘图语言 Asymptote, 包括安装与配置, 2D 绘图与 3D 绘图, 是笔者学习 asy 语言的一些经验总结, 讲解略显简略, 特别是其中的 2D 绘图部分还没有充分展开, 留待他日有机会再完成, 本文档也因此冠名为 asy 笔记而不是通常的教程或讲义, 只是希望对新手有一些帮助.

本文档的是在 C_T_EX 论坛的各位版主的支持和帮助下完成的, 包括 milksea(刘海洋), LiYanrui(李延瑞), Neals(陈之初), zpxing(邢兆鹏) 等等, 特别感谢 milksea, 是他帮忙设计了本文档的大部分格式.

最后要感谢法国的 Ivaldi 和 Marris, 本文档中的一些例子是取自他们的 asy 网页, 还要感谢 METAPOST 的 m3D 宏包作者, 法国的 Anthony Phan, 本文档中不少精彩的 3D 例子是从这个宏包提供的例子翻译改进得到的.

本文档是 asy4cn 项目的一部分, 可以在 <http://code.google.com/p/asy4cn/downloads/list> 找到更多的学习资料以及相应的更新.

目录

| | |
|-----------------------------------|-----------|
| 第一章 Asymptote 简介, 安装与配置 | 1 |
| 1.1 Asymptote 简介 | 1 |
| 1.2 Asymptote 的安装 | 2 |
| 1.2.1 Windows 用户的安装 | 2 |
| 1.2.2 Linux 用户的安装 | 3 |
| 1.3 Asymptote 的配置 | 7 |
| 1.4 测试 | 11 |
| 1.5 中文标签 | 13 |
| 1.5.1 XeLaTeX 方案 | 13 |
| 1.5.2 ConTeXt-LuaTeX 方案 | 14 |
| 1.5.3 CJK 方案 | 16 |
| 1.6 如何编辑 asy 文件 | 16 |
| 1.6.1 用 emacs 编辑器 | 16 |
| 1.6.2 用 vim 编辑器 | 17 |
| 1.6.3 用 SciTE 编辑器 | 18 |
| 1.6.4 用 NotePad++ 编辑器 | 19 |
| 1.7 卸载 | 22 |
| 第二章 2D 绘图 | 25 |
| 2.1 基本的绘图命令 | 25 |
| 2.2 画线段和点 | 26 |
| 2.3 画虚线 | 26 |
| 2.4 画箭头 | 27 |
| 2.5 画线段按比例的分点 | 29 |
| 2.6 画过若干个点的曲线 | 31 |
| 2.7 变量与函数 | 32 |
| 2.8 画正多边形 | 33 |
| 2.9 画圆弧 | 33 |
| 2.10 画直角记号 | 34 |

| | |
|---|-----------|
| 2.11 自定义函数 | 35 |
| 2.12 画函数图像 | 35 |
| 2.13 参数曲线和隐函数表达的曲线 | 37 |
| 2.14 画阴影 | 38 |
| 2.15 子图 | 39 |
| 2.16 裁剪命令 clip | 40 |
| 2.17 unfill 命令 | 42 |
| 2.18 奇偶法则填颜色 | 42 |
| 2.19 变换 | 42 |
| 2.20 简单的编程 | 44 |
| 2.21 与向量 pair 有关的一些函数 | 48 |
| 2.22 Bezier 曲线的原理及与 path 有关函数 | 49 |
| 2.23 数组 | 53 |
| 2.24 subfigure | 53 |
| 2.25 graph 宏包 | 54 |
| 2.26 极坐标 | 56 |
| 2.27 数项级数表示的函数 | 56 |
| 2.28 分段函数 | 58 |
| 2.29 画线程图 | 59 |
| 2.30 读入数据绘图 | 59 |
| 2.31 各种循环语句 | 62 |
| 2.32 弯曲路径上标标签 | 63 |
| 2.33 随机 | 64 |
| 2.34 模拟徒手绘图的效果 | 65 |
| 2.35 分形图 | 66 |
| 第三章 3D 绘图 | 71 |
| 3.1 Asymptote 三维作图概述 | 71 |
| 3.2 预览 3D 输出 | 71 |
| 3.3 关于 3D 投影 | 73 |
| 3.3.1 透视投影 | 73 |
| 3.3.2 正交投影 | 74 |
| 3.3.3 斜投影 | 75 |
| 3.4 3D 的数据类型 | 76 |
| 3.5 3D 曲线 | 77 |
| 3.6 Bezier 曲面片 | 80 |
| 3.7 画参数曲面 | 84 |
| 3.8 3D 字母, 坐标轴, 刻度和栅格 (grid) | 87 |

| | | |
|------------|----------------------------|------------|
| 3.9 | 3D 变换 | 94 |
| 3.10 | 着色处理和改变背景颜色 | 97 |
| 3.11 | 柱状图形与立体字体 | 103 |
| 3.12 | 球面坐标 | 106 |
| 3.13 | 旋转体 | 110 |
| 3.14 | 曲面上画曲线, 利用 lift 函数画两个曲面的交线 | 118 |
| 3.15 | 画等高线 | 123 |
| 3.16 | 画曲面的局部 | 128 |
| 3.17 | 直线段与平面的交点, 曲线与曲面的交点 | 133 |
| 3.18 | 多视角看同一个图形 | 135 |
| 3.19 | tube 函数 | 136 |
| 3.20 | contour3 函数 | 138 |
| 3.21 | 常见的二次曲面 | 140 |
| 3.21.1 | 单叶双曲面 | 141 |
| 3.21.2 | 双叶双曲面 | 143 |
| 3.21.3 | 椭球面 | 144 |
| 3.21.4 | 椭圆抛物面 | 145 |
| 3.21.5 | 双曲抛物面 | 146 |
| 3.21.6 | 双曲柱面, 椭圆柱面, 抛物柱面 | 147 |
| 3.22 | 画线框 | 147 |
| 3.23 | 用 extrude 函数画直纹面 | 152 |
| 3.24 | 画复数函数的图像 | 153 |
| 3.25 | Bezier 曲面的构造 | 155 |
| 3.26 | 各种多面体 | 158 |
| 3.27 | 球面多边形, 足球与 C60 | 160 |
| 3.28 | 曲面上的向量场 | 165 |
| 3.29 | 离散数据的可视化 | 167 |
| 3.30 | NURBS | 170 |
| 3.31 | Sierpinski Sponge | 172 |
| 3.32 | 一些示例 | 180 |
| 3.32.1 | 立方体堆成的三视图 | 180 |
| 3.32.2 | Viviani 立体和 Viviani 曲线 | 181 |
| 3.32.3 | Mobius 带原理的演示 | 183 |
| 第四章 | 一些论题 | 185 |
| 4.1 | 把 asy 代码嵌入 tex 文件的编译方案 | 185 |
| 4.1.1 | 常规方案 | 185 |
| 4.1.2 | 用 latexmk 脚本的方案 | 185 |

| | | |
|-----|----------------------------------|-----|
| 4.2 | 动画 | 186 |
| 4.3 | 做 3D 动画 | 188 |
| 4.4 | 做 3D 幻灯 | 191 |
| 4.5 | 做动画幻灯 | 193 |
| 4.6 | 嵌入多媒体 | 195 |
| 4.7 | 用 Asymptote 添加图片中的字母标签 | 196 |
| 4.8 | 与 Mathematica 结合 | 197 |

第一章 Asymptote 简介, 安装与配置

1.1 Asymptote 简介

下图是用 Asymptote 绘图语言生成的, 它是 Asymptote 的 3D 徽标, 如果用 Adobe 的 pdf 阅读器 (9.0 以上版本) 打开本文档, 那么可以用鼠标对它进行一系列操作, 这是 Asymptote 1.44 版本开始支持的功能.

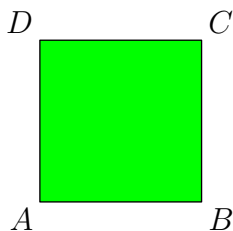


Asymptote 源于 2002 年加拿大 Alberta 大学的一个项目, 由 Alberta 大学的 Bowman 和 Toronto 大学的 Hammerlindl 发起. 当初的目的是希望把大名鼎鼎的科学绘图语言 METAPOST 存储实数的方式由低精度的定点数改为浮点数, 以便更适应数据绘图的需要, 于是他们决定重新写一个绘图引擎. 经过半年的工作, 终于可以画出正弦曲线, 完成了 `draw` 这个功能.

在接下去很快完成了在封闭区域填充的 `fill` 功能. 而利用 L^AT_EX 的 `\includegraphics` 功能把 T_EX 生成的标签标到图上面的 `label` 功能也在随后的几个月完成了.

```
size(0,3cm);  
fill(unitsquare,green);  
draw(unitsquare);  
label("$A$", (0,0),SW);  
label("$B$", (1,0),SE);
```

```
label("$C$", (1,1),NE);
label("$D$", (0,1),NW);
```



随后, Asymptote 的第三作者 Prince 加入, 他提供了把 `asy` 代码嵌入 `tex` 文件中的办法.

2004 年 11 月 4 日, Asymptote 在 sourceforge.net 发布了第一个版本 0.51 版本, 2005 年 8 月 13 日放出的 0.84 版本开始支持 Windows. 2007 年 10 月 11 日的 1.34 版本开始支持代码与鼠标结合的绘图方式. 从 2008 年下半年更开始支持直接生成 PRC 真 3D 格式以及基于 OpenGL 的 3D 渲染, 而 PRC 真 3D 格式嵌入 pdf 后的效果就是我们前面的 Asymptote 的示例, 这使得 Asymptote 一跃成为非常引人瞩目的科学绘图语言.

在 2010 年 6 月后的版本非常有效加快了 PRC 真 3D 的显示速度, 因此, 结合 Asymptote 提供的幻灯模式制作的 pdf 成为显示真 3D 非常实用的工具. 另外, `asy` 的 Windows 版本效率获得了极大的提高, 那个让人诟病的 384M 内存限制也去掉了, 因此, 在与 TeX 结合的绘图工具中, Asymptote 显出极大的优势.

1.2 Asymptote 的安装

在安装 Asymptote 之前, 我们的系统应该事先装有一个 TeX 系统, 使得起码能够在其上使用 L^AT_EX, 推荐安装针对跨平台的 TeXLive 的最新版. 然后到 Asymptote 官方网页 <http://asymptote.sourceforge.net/> 的 Download 链接下载相应于各个平台的最新的版本, 下面分别针对不同的平台说说它们的安装办法.

1.2.1 Windows 用户的安装

Windows 用户直接下载 `asymptote-x.xx-setup.exe` 安装, 其中 `x.xx` 是版本号.

Windows 用户除了安装 TeX 以外, 还应该安装 GPL Ghostscript, 见 <http://sourceforge.net/projects/ghostscript/>. `asy` 要依赖于它来把 ps 解释成 pdf, 以及把 pdf 解释成 ps(用于把 xelatex, pdflatex, ConTeXt 所生成的 pdf 字体文件转成 ps 文件, 以便利用这个 ps 文件生成 3D 图形中的字母标签)

接下去需要安装 `gsview` 用于预览那些生成的 `eps` 文件. 见 <http://www.cs.wisc.edu/~ghost/gsview/>. 也可以安装一个免费的 `eps` 阅读器 (这个在 TeXLive 里面也已经有了), 见 <http://psview.sourceforge.net/> 我们可以把它解压到 `c:\Program Files` 目录下, 然后在 `C:\Documents and Settings\您的用户名\.asy` 目录下的 `config.asy` 中加入 (其中 `x.xx` 是版本号)

```
import settings;
psviewer="c:\Program Files\psview-x.xx\psv.exe";
```

默认安装目录中的 C:\Program Files\Asymptote\examples 里面丰富的例子是学习 Asymptote 的标准材料。

接下去双击桌面上 Asymptote 的图标, 那么就会进入 asy 命令行的等待状态, 我们可以测试一下我们的安装, 试试输入一些绘图命令, 观察出来的结果. (用 quit, q 或 exit 退出)

```
Welcome to Asymptote version 2.08 (to view the manual, type help)
> size(200);
> import three;
> draw(unitsphere,red);
> erase();
> draw(unitcircle,blue);
>
```

我们画了一个红色的球和一个蓝色的圆. 其中的 `erase()` 函数擦除之前的设置重新画另外一幅图. 如果能正确弹出一个基于 OpenGL 的窗口显示球 (如果想导出那个窗口的图形, 用 `e` 快捷键), 然后弹出一个 `gsview32.exe` 的预览窗口显示圆, 那么我们的安装基本上没有大问题. 接下去我们应该参考第 7 页的相应章节进一步配置, 特别的, 我们要注意 `asy` 所在路径加入到 `Path` 这个环境变量中的这个问题.

我们也可以把代码保存为一个以 `asy` 为扩展名的文件 (比如 `test.asy`), 然后双击该文件生成相应的图形文件. 不过我们还是建议按照第 8 页的方法右击文件进入 DOS 命令行, 用

```
asy -V test.asy
```

命令编译, 因为这样可以看到详细的出错信息.

更进一步的, 我们需要一个好的编辑器编辑 `asy` 代码, 并且配置好编辑器调用 `asy` 编译当前的代码, 详细的讨论见第 16 页.

1.2.2 Linux 用户的安装

安装可执行的发行版

针对 Linux 的系统的有 `tgz` 和 `RPM` 两种可执行发行版. Fedore Core 的用户可以用

```
yum --enablerepo=rawhide install asymptote
```

安装和更新为最新的 Asymptote 发行版本.

也可以安装 Linux `i386 tgz` 发行版. 下面以 Ubuntu 操作系统为例. 进入 `asymptote-x.xx.i386.tgz` 所在目录, 然后执行 (其中 `x.xx`要改为相应的版本号):

```
sudo tar -C / -zxvf asymptote-x.xx.i386.tgz
```

上述命令是先利用 `-C /` 切换到根目录 `/`, 然后再利用 `tar -zxf` 解压. 最后会把可执行文件 `asy` 安装到 `/usr/local/bin` 目录下, 把一些共享文件 (比如 `asy` 要读入的宏包) 放在 `/usr/local/share/asympote` 目录下. `/usr/local/share/doc/asympote` 目录下包含的 `asympote.pdf` 以及各种例子的 `examples` 目录, 还有 `FAQ` 等等, 这些都是 Asymptote 学习的重要资料.

由于 `asy` 是安装到 `/usr/local/bin` 目录中, 因此为了保证系统用的是这个路径中的 `asy` 而不是 `texlive` 中的 `asy` (它通常在 `/usr/local/texlive/2010/bin/i386-linux` 等目录下), 我们可以在 `~/.profile` 文件里面把 `/usr/local/bin` 放到最前面, 例如把下面代码加到 `~/.profile` 后面, 注销以后再登入.

```
PATH=/usr/local/bin:/usr/local/texlive/2010/bin/i386-linux:$PATH
export PATH
```

接下去我们要把刚刚解压后的 `/usr/share/texmf/tex/latex/asympote` 文件夹移动到您的 `TeX` 系统的相应目录下, `TeXLive` 可以是 `/usr/local/texlive/texmf-local/tex/latex`, 然后再执行 `texhash`. 当然, 可能要先 `export` 一下 `tex` 那些执行文件所在的路径, 比如 `Ubuntu` 下的 `TeXLive2009` 可能要先执行如下命令:

```
export PATH=/usr/local/texlive/2009/bin/i386-linux:$PATH
texhash
```

`Linux` 的用户此时可以打开一个终端, 输入 `asy`, 这时应该弹出类似于

```
Welcome to Asymptote version 2.08 (to view the manual, type help)
>
```

的等待状态. 如果有出错信息, 说明缺少某些库文件, 这时应该装上相应的库文件或链接, 可以参考 `apt-get install asympote` 里面的依赖关系把所需要的库装上. 特别地, 我们需要 `freeglut` 这个基于 `OpenGL` 的库, 如果没有, 会提示缺少 `libglut.so.3`, 这时在 `Ubuntu` 简单地用 `apt-get install freeglut3-dev` 装上就可以. 其他的库我们可以根据出错信息安装. 另外, 特别要注意的是, 如果提示缺少 `libtinfo.so.5`, 那么可能要执行

```
sudo ln -s /lib/libncurses.so.5 /usr/lib/libtinfo.so.5
```

其中的 `/lib/libncurses.so.5` 是 `Ubuntu` 里面库文件的路径. 类似的, 如果 `libglut.so.3` 是在 `/usr/local/lib/libglut.so.3` 那里, 那么执行

```
sudo ln -s /usr/local/lib/libglut.so.3 /usr/lib/libglut.so.3
```

如果提示缺少 `libreadline.so.6`, 我们可以执行

```
sudo ln -s /lib/libreadline.so.5 /usr/lib/libreadline.so.6
```

其他库的依赖类似地处理, 例如新近的版本需要 `libfftw3.so.3`, 我们可以下载 `fftw` 编译源代码, 不过要注意要用

```
./configure --enable-shared
```

来生成库文件, 然后执行

```
sudo ln -s /usr/local/lib/libfftw3.so.3 /usr/lib/libfftw3.so.3
```

在 Linux 下 Asymptote 默认是采用轻量级的 gv 预览生成的 eps 文件的. 对于生成的 pdf 文件, 默认是用 acroread 预览的, 因为一些与 pdf 有关的 PRC 真 3D 效果以及动画效果暂时只能用它来读.

如果解决不了 Asymptote 所依赖的库的关系, 建议参考后面所说的编译源代码方法来安装, 特别是 64 位操作系统.

从 svn 源代码或源代码包编译安装

请事先装好 GNU make 等编译工具, 对于 debian/ubuntu 的用户, 可以执行

```
sudo apt-get install subversion autoconf gcc g++ bison flex make texinfo
```

然后安装相关的库文件, ubuntu 的用户再装

```
sudo apt-get install libzip-dev
```

debian 的用户装

```
sudo apt-get install zlib1g-dev
```

建立编译环境时, 在 Ubuntu 下也可以简单地用

```
sudo apt-get build-dep asymptote
```

然后参考那个列表自己手动安装除了 texlive 以外的依赖的库都装上.(因为不建议用装源里面的那个 TeX 系统).

接下去下载最新的 svn 版本, 即执行

```
mkdir asymptote_svn && cd asymptote_svn
```

```
svn co http://asymptote.svn.sourceforge.net/svnroot/asymptote/trunk/asymptote
```

编译时默认是用系统的 Boehm garbage collector(即垃圾回收器 gc), 不过推荐到 http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_source/ 下载最新的 gc-x.x.tar.gz 放到 asymptote 目录下. 比如可以执行

```
cd asymptote && ./autogen.sh
```

```
wget http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_source/gc-7.1.tar.gz
```

接下去要装 freeglut, 基于 OpenGL 的 3D 渲染(render) 需要这个库. 建议下载 <http://freeglut.svn.sourceforge.net/viewvc/freeglut/trunk/freeglut/freeglut.tar.gz> 到 asymptote 目录下, 先用

```
sudo apt-get build-dep freeglut
```

建立编译环境, 然后执行

```
tar -zxf freeglut.tar.gz
cd freeglut
sh autogen.sh
./configure --prefix=/usr
sudo make install
cd ..
```

现在就可以开始编译 Asymptote 了.

```
./configure
make all
sudo make install
```

以后可以随时如下更新, 从而用上最新的 svn 版本.

```
cd ~/asymptote_svn/asymptote
svn update
make all
sudo make install
make clean
```

类似地, 我们可以从源代码安装最新的稳定版, 步骤与上面大同小异, 下载 `asymptote-x.xx-src.tgz`, 然后执行 (其中 `x.xx` 是版本号).

```
tar -zxf asymptote-x.xx.src.tgz
cd asymptote-x.xx
wget http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_source/gc-7.1.tar.gz
下载下列文件到当前目录
http://freeglut.svn.sourceforge.net/viewvc/freeglut/trunk/freeglut/freeglut.tar.gz
tar -zxf freeglut.tar.gz
cd freeglut
sh autogen.sh
./configure --prefix=/usr
sudo make install
cd ..
./configure
make all
sudo make install
```


1.3 Asymptote 的配置

装好 Asymptote 以后, 一般还要进行一下配置, 才能让可执行文件 `asy` 正常的工作.

首先, 我们要保证要有一个相对较新的 $\text{T}_\text{E}\text{X}$ 系统. 适当时候我们可以更新一下宏包. 对 Asymptote 来说, 通常需要最新的 `movie15.sty` 和 `animate.sty`, 请分别下载下面两个文件, 即 <http://www.ctan.org/tex-archive/macros/latex/contrib/movie15/movie15.sty> 以及 <http://www.ctan.org/tex-archive/macros/latex/contrib/animate/animate.sty> 然后替换掉 $\text{T}_\text{E}\text{X}$ 系统里面相应文件, 必要时候执行一下 `texhash` 命令更新数据库.(如果是用 `xelatex`, 那么要采用适用于 `xelatex` 的 `movie15.sty`, 见第 13 页.)

`movie15.sty` 用于把 PRC 真 3D 嵌入到 pdf 中, 见前面 Asymptote 徽标的效果. `animate.sty` 用于把 Asymptote 生成的一系列 eps 或 pdf 组合成一个嵌入 pdf 中的动画. 效果如下图所示.

另外, 如果我们希望用 Asymptote 把 `asy` 代码嵌入 `tex` 代码中编译, 或者利用 Asymptote 做幻灯, 做动画, 那么还需要把默认安装目录 `C:\Program Files\Asymptote` 中的 `asymptote.sty`, `asycolors.sty` 和 `ocg.sty` 作为一个宏包安装到 $\text{T}_\text{E}\text{X}$ 系统当中, 方法与装其他一些宏包一样, 即找到本地诸如 `texmf-local/tex/latex` 目录, 新建一个 Asymptote 的文件夹, 然后把刚才那些 `.sty` 文件复制到里面, 再执行 `texhash`.

有时候, 我们希望生成 EPS 和 PDF 以外的格式, 比如利用 Asymptote 生成 GIF 和 MPEG 动画, 这时需要 ImageMagick 这个软件, 见 <http://www.imagemagick.org/script/binary-releases.php> Linux 的用户还需要安装 `ffmpeg`, Ubuntu 下用

```
sudo apt-get install ffmpeg
```

安装, 然后可以测试 `asymptote` 的 `examples` 目录下, `animations` 文件夹里面的 `slidemovies.asy` (`wheel.asy` 要在同一目录中)

为了使用 `xasy` 这个鼠标绘图功能, 我们需要先后安装 Python 以及相应的图形库.

<http://www.python.org/ftp/python> 选择安装 python 的版本 (比如 2.6.2 版本的 `python-2.6.2.msi`) <http://www.pythonware.com/products/pil/> 下载对应于 Python 版本的 PIL 库, 比如 `PIL-1.1.7.win32-py2.6.exe`.

Windows 用户建议进行如下的配置, 因为很多时候需要在 `asy` 命令后面加一些参数来生成所想要的格式, 而且在命令行执行任务可以看到出错信息, 方便我们进一步测试我们的配置是否成功.

先把 `asy.bat` 和 `asy.exe` 所在目录加入环境变量中的 `PATH` 中, 具体做法是:

1. 单击 开始
2. 右击 我的电脑
3. 选择 属性
4. 单击 高级
5. 单击 环境变量
6. 双击系统变量下的 `Path` 这个环境变量, 看看这个变量值里面是否有 `C:\Program Files\Asymptote;` 这一项, 而且是否在 `TeXLive` 路径名的前面 (通常是类似于 `C:\texlive\2010\bin\win32;`), 如果不是, 请把 `C:\Program Files\Asymptote;` 添加到变量值的最前面. 这样才能保证我们调用的是最新的 `asy`.

接下去在右键加入进入 DOS 命令行. 方法是把下面代码保存为 `asy.reg`, 然后双击, 那么就会在右键增加 进入命令行, 我们就可以方便地执行 `asy` 命令了.

Windows Registry Editor Version 5.00

```
[HKEY_CURRENT_USER\Software\Classes]
```

```
[HKEY_CURRENT_USER\Software\Classes\*]
```

```
[HKEY_CURRENT_USER\Software\Classes\*\shell]
```

```
[HKEY_CURRENT_USER\Software\Classes\*\shell\ 进入命令行]
```

```
[HKEY_CURRENT_USER\Software\Classes\*\shell\ 进入命令行 \command]
```

```
@="cmd"
```

```
[HKEY_CURRENT_USER\Software\Classes\Folder]
```

```
[HKEY_CURRENT_USER\Software\Classes\Folder\shell]
```

```
[HKEY_CURRENT_USER\Software\Classes\Folder\shell\ 进入命令行]
```

```
[HKEY_CURRENT_USER\Software\Classes\Folder\shell\ 进入命令行 \command]
```

```
@="cmd /k cd \"%1\""
```

在 Windows 下, Asymptote 的配置文件 `config.asy` 通常是 `C:\Documents and Settings\ 您的用户名\.asy` 目录下的 `config.asy`, Linux 下是 `~/.asy/config.asy` (如果没有, 都请手工新建一个), 里面的内容可以仿照下面的格式来写, 并请针对您的系统做一些修改. 特别是 `//` 注释号部分那几行一般情况下是可以不用写上去的.

```
import settings;
gs="gs";
psviewer="gv";
pdfviewer="acroread";
python="";
//glOptions="-indirect";
//dir="";
//interactiveView=true;
//batchView=true;
```

在 Windows 下的配置文件可以仿照如下的来写 (通常情况下是不需要写的, 另外也请按照您系统里面的路径作适当修改)

```
import settings;
gs="C:\Program Files\gs\gs9.00\bin\gswin32c.exe";
psviewer="C:\Program Files\Ghostgum\gsview\gsview32.exe";
pdfviewer="C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe";
python="";
//dir="";
//interactiveView=false;
//batchView=false;
```

上述各项选项的涵义如下:

`gs` // PostScript 的解释程序, 预览 eps 文件以及要把 eps 文件转成 pdf 时需要调用它.

`psviewer` // eps, ps 等 PostScript 文件的查看程序.

`pdfviewer` // PDF 文件的查看程序.

`python` // xasy 调用的 python 程序.

`glOptions` // 基于 OpenGL 的 3D 渲染时显卡驱动选项.

`dir` // asy 的搜索路径, 该路径通常用于放置 asy 的一些宏包.

`interactiveView` // 在命令行进行交互式绘图时是否预览.

`batchView` // 在对 asy 文件编译时是否进行预览.

请根据系统修改路径, 特别注意的是: Windows 下 `gs` 应该是 `gswin32c.exe` 而不是 `gswin32.exe`. 另外, 建议最好选择 Gsview 里面的 *fit to window to page* 和 *eps clip*, 前面一个的作用, 使得窗口自动调整和图片一样大; 后面一个的作用, 使得对于的 eps 图片部分被剪裁掉.

其他的选项一般都不用设, asy 会自动探测的, 因为它们一般都已经写入了系统的环境变量 `PATH` 中了, 它们是:

```

texpath // 指系统里面的 TeX 的可执行文件所在目录

texcommand // 指 asy 调用的 TeX 引擎

dvips // dvips 命令所在目录

convert // 转换图像格式的 ImageMagick 的 convert 程序所在路径

display // 显示 png 格式文件程序所在路径

animate // 显示 gif 格式文件程序所在路径

xasy // 鼠标绘图的 xasy 所在路径

```

我们可以顺便介绍一下 asy 的搜索路径. 首先是当前目录, 其次是命令行的 `-dir=""` 参数所指定的目录, 接下去是 `./asy` 文件夹, 然后是 asy 的安装目录. 因此, 我们不能把一些与 asy 已有宏包名字相同的一些 asy 文件放在上述目录中, 否则会出错, 而另一方面, 我们可以把一些经常用到的自定义宏包方在上述目录中, 然后用 `autoimport` 在 `~/.asy/config.asy` 中设置, 具体应用见 ConTeXt 的配置 15

另外, 在 Linux 下我们用 acroread 预览 asy 编译出来的 pdf 可以不用关掉那个 acroread, 只需把 `/usr/local/share/asymptote` 下的 `reload.js` 复制到 `~/.adobe/Acrobat/8.0/JavaScripts/` 目录下, 然后在 `~/.asy/config.asy` 中加入

```

import settings;
pdfreload=true;
pdfreloadOptions="-tempFile";

```

那么以后再编译那个 asy 文件会自动刷新 pdf 预览.

最后, 列出 Asymptote 的一些常用的选项, 这些选项大部分既可以在命令行下作为参数用, 当把 `-` 去掉以后也可以写在那个 `./asy/config.asy` 文件里面, 另外, 如果把 `-` 换成 `-no`, 那么就代表对应的否定选项.

| | |
|-------------------------------|-----------------------|
| <code>-V, -View</code> | 预览输出, 只对命令行有效 |
| <code>-batchView</code> | 批处理地预览输出 [true] |
| <code>-interactiveView</code> | 交互式地预览 [true] |
| <code>-fitscreen</code> | 3D 渲染的图像适合屏幕. [true] |
| <code>-grey</code> | 把所有的颜色转成灰色. [false] |
| <code>-k</code> | 保留执行编译过程中的文件. [false] |
| <code>-maxtile pair</code> | 控制 3D 渲染的窗口的尺寸 [0,0] |

| | |
|----------------------|---|
| -f -outformat format | 按照这个指定去输出文件格式 |
| -pdfreload | 在 pdf 阅读器中自动刷新. <code>[false]</code> |
| -prc | 嵌入 PRC 真 3D 格式到 pdf 文件中. <code>[true]</code> |
| -render n | 以 n 像素渲染 3D 图形, 0 代表生成 2D 向量图, -1 代表自动. |
| -auto3D | 自动激活基于 OpenGL 的 3D 渲染屏幕显示. <code>[true]</code> |
| -tex engine | T _E X 引擎, 可以是 <code>latex</code> , <code>pdflatex</code> , <code>xelatex</code> , <code>context</code> , <code>tex</code> , <code>pdftex</code> , <code>none</code> <code>[latex]</code> |
| -toolbar | 在真 3D 的 pdf 中出现工具条. <code>[true]</code> |
| -wait | 等待子进程完成才退出. <code>[false]</code> |
| -autoimport | 自动加载其后字符串所示的文件, 该文件通常放在 asy 的默认搜索路径中. |
| -multiline | 采用该选项可以在命令行一次粘贴多行代码, 这个方便于迅速测试代码. <code>[false]</code> |

举一个例子, 如果我们想让 asy 输出 pdf 文件, 并且用 xelatex 作为 T_EX 引擎, 那么可以用下面几种办法中的任何一种.

可以在命令行下用

```
asy -f pdf -tex xelatex filename.asy
```

可以在 asy 文件开头写上

```
import settings;
outformat="pdf";
tex="xelatex";
```

或者更简洁的

```
settings.outformat="pdf";
settings.tex="xelatex";
```

可以在上面的 config.asy 文件里面加上

```
outformat="pdf";
tex="xelatex";
```

1.4 测试

找到前面提及的 examples 目录, 找到里面的比如 venn.asy 和 epix.asy 以及 latexusage.tex, 然后对它进行测试. 其中的 latexusage.tex 是把 asy 文件嵌入 tex 文件的示例. 详细的讨论见第 185 页.

执行

```
asy -V venn.asy
```

将会弹出一个 `eps` 文件阅读器的窗口.

执行

```
asy -V -f pdf venn.asy
```

将会弹出一个 `pdf` 文件阅读器的窗口.

执行

```
asy -V epix.asy
```

将会弹出一个 OpenGL 的窗口, 用鼠标可以对图形进行操作.

执行

```
asy -V -f pdf epix.asy
```

应该弹出一个 Adobe 的 pdf 阅读器, 打开的是嵌入 pdf 中的真 3D 图形, 同样可以用鼠标对图形进行各种操作.

执行

```
asy -V -f pdf -noprc epix.asy
```

应该现弹出一个 OpenGL 的窗口, 我们把窗口最大化以后, 用 `q` 键可以退出这个窗口, 接着应该会接着弹出一个 pdf 阅读器, 这个就不再是前面的真 3D 图形, 而是把刚才 OpenGL 看到的图形”拍摄”出来的结果.

执行

```
asy -noV -f pdf -noprc epix.asy
```

这时弹出一个窗口, 不显示任何东西, 不过在弹出窗口后, 会在当前目录生成了与前面命令一样的 pdf. 如果图形错乱或出现其他异常现象, 请尝试使用

```
asy -noV -f pdf -noprc -maxtile="(512,512)" epix.asy
```

以及参考第三章 3D 部分的论述.

执行

```
asy -noV -f pdf -render=4 epix.asy
```

一旦设了 `-render=4`, 那么该 pdf 的”封面”可以被任意的 pdf 阅读器看到, 如果是 Adobe 9.0 以上的阅读器, 那么可以进一步激活 PRC 真 3D.

依次执行

```
pdflatex latexusage.tex
asy latexusage-*.asy
pdflatex latexusage.tex
```

1.5 中文标签

我们知道, 现在中文的三大处理方案是 CJK, XeTeX, LuaTeX. 自 1.75 版本 (2009 年夏季) 以来, Asymptote 对它们都有很好的支持!

1.5.1 XeLaTeX 方案

目前 1.75 及其以后版本 xelatex 与 asy 的配合已经可以标注 2D 和 3D 图形的中文标签了. 其中 3D 标签的主要原理是以 XeLaTeX 作为引擎, 生成中文字体的 pdf 文件, 然后再利用 GhostScript 生成一个包含中文字体轮廓信息的 ps 文件, 接下去就与其他英文标签一样的原理, 利用这些字体路径得到 3D Bezier 曲面片, 从而实现在 3D 图形中标中文. 但要注意, 如果要在 3D 图形中用 xelatex, 并且生成嵌入 PRC 真 3D 的 pdf, 那么要采用同时适用于 dvipdfmx 和 pdflatex 的 movie15_dvipdfmx.sty, 我们可以到 <http://asymptote.svn.sourceforge.net/viewvc/asymptote/trunk/asymptote/patches/> 下载 (或者下载源代码包 asymptote-*.src.tgz, 解压后在 patches 找到它) 然后把它重命名为 movie15.sty, 并且替换掉系统里面的 movie15.sty, 在必要的情况下可能要相应的更新 asymptote.sty 到最新的版本.)

请分别测试下面简单的例子. 注意要用 UTF-8 编码保存代码为 test.asy, 并且 T_EX 系统里面要装有 ctex 宏包以及 xeCJK 中文宏包以及相应的字体 (用 `fc-list :lang=zh` 查看是否有相应的字体) 直接用 `asy -V test.asy` 编译.

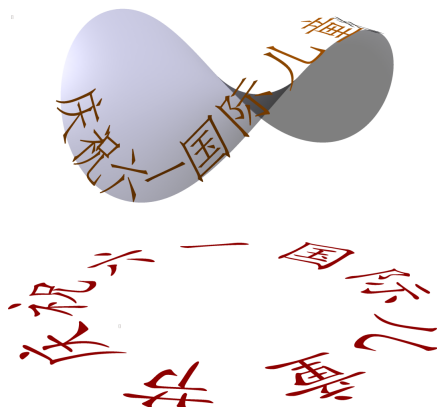
```
//需要 Windows 的 SimSun 等字体或 Adobe 的 Adobe Song Std 等字体.
settings.tex="xelatex";
usepackage("ctex");
//usepackage("ctex","adobefonts");
label("中文");
```

```
settings.tex="xelatex";
//settings.prc=false;
texpreamble("\usepackage{xeCJK}");
texpreamble("\setCJKmainfont{SimSun}");
//texpreamble("\setCJKmainfont{Adobe Song Std}");
import three;
size(200);
draw(unitbox,red);
label("顶点  $A$ ",(1,1,1),2Z);
```

1.5.2 ConTeXt-LuaTeX 方案

主要原理是以 ConTeXt 作为引擎, 通过 LuaTeX 生成中文字体, 然后再生成一个包含中文字体轮廓信息的 ps 文件, 接下去就与其他英文标签一样的原理, 利用这些字体路径得到 3D Bezier 曲面片, 从而实现在 3D 图形中标中文.

用 zhfonts.tex 的做法



```
settings.tex="context";
texpreamble("\usetypescriptfile[zhfonts]");
texpreamble("\usetypescript[myfont]");
defaultpen(font("myfont")+linewidth(1pt));
import labelpath3;
import graph3;
size(200);
path3 g=(1,0,0)..(0,1,1)..(-1,0,0)..(0,-1,1)..cycle;
path3 g2=shift(-Z)*reverse(unitcircle3);
string txt1="\tt{ 庆祝六一国际儿童节 }";
string txt2="\ss{ 庆祝六一国际儿童节 }";
draw(surface(g),paleblue+opacity(0.5));
draw(labelpath(txt1,subpath(g,-0.5,reltime(g,0.3)),angle=-90),orange);
draw(labelpath(txt2,subpath(g2,0,3.9),angle=180,optional=rotate(-70,X)*Z),red);
```

首先要有比较新的 ConTeXt Minimals. 这里是针对 Linux 用户去说明, Windows 的用户类似处理.

假定我们用那些 Adobe 字体, 那些文件放在 /usr/share/fonts/adobe 目录下, 那么我们在 ~/context/tex/setuptex 里面加入一句 export OSFONTDIR="/usr/share/fonts/adobe", 然后执行 source ~/context/tex/setuptex(可以把这行命令写入 ~/.bashrc 里面) 来设置 context 环境, 然后执行 mtxrun --script fonts --reload 来更新字体信息.

其中我们需要的 zhfonts.tex 在 <http://code.google.com/p/way2ctx/downloads/list> 的 obsolete.tar.gz 里面, 请把它放在当前目录下, 或者放到比如 ~/context/tex/texmf-local/tex/context/third/ 目录下, 然后执行 `context --generte`(或其他命令) 并且相应地修改里面的字体设定,

然后我们就可以在 `source ~/context/tex/setuptex` 设置好的环境下在命令行编译上述例子了.

如果觉得每次都要修改相应的 asy 文件才能编译比较麻烦, 那么可以在 `/.asy/` 建立一个文件 context.asy, 里面的内容是

```
settings.tex="context";
texpreamble("\usetyescriptfile[zhfonts.tex]");
texpreamble("\usetyescript[myfont]");
defaultpen(font("myfont"));
```

然后在 `/.asy/config.asy` 里面加入

```
import settings;
autoimport="context";
```

那么 asy 就会自动去读入那个 `/.asy/` 目录下的 context.asy.

用 simplefonts 的做法

假设 ConTeXt Minimals 被安装在 ~/context 目录, 那么就将 Simplefonts 解压后所得目录放置于 ~/context/tex/texmf-local/tex/context/third 目录下, `context --generte` 更新数据库.(如果在下载安装 ConTeXt Minimal 时用了 `-extras=all` 等选项, 那么就已经装好了, 不用手工下载)

这次是在 `~/context.asy` 里面加入

```
settings.tex="context";
texpreamble("
\usemodule[simplefonts]
\setupsimplefonts

\setmainfont[TeXGyrePagella]
\setsansfont[TeXGyreHeros]
\setmonofont[TeXGyreCursor]

\setcjkmainfont
  [AdobeSongStd]
  [regularfont={* Light},
  italicfont={AdobeKaitiStd Regular},
  boldfont={AdobeHeitiStd Regular},
  bolditalicfont={AdobeHeitiStd Regular}]
```

```

\setcjkssansfont
    [AdobeKaitiStd]
    [boldfont={AdobeHeitiStd Regular},
     bolditalicfont={AdobeHeitiStd Regular}]
\setcjkmonofont
    [AdobeFangsongStd]
    [boldfont={AdobeHeitiStd Regular},
     bolditalicfont={AdobeHeitiStd Regular}]
");

defaultpen(font("simplefonts"));

```

要注意的我们这里才用了 Adobe 的几款 otf 字体, 我们可以根据自己系统理面的情况修改字体设置. 在 `~/.asy/config.asy` 里面还是用 `autoimport` 自动导入那个 `context.asy`

```

import settings;
autoimport="context";

```

然后就可以测试前面的例子了.

1.5.3 CJK 方案

假定已经安装了 TeXLive, 里面有中文字体 gkai, 否则要把 gkai 换成自己生成的字体文件名.

```

texpreamble("\usepackage{CJK}
\AtBeginDocument{\begin{CJK*}{UTF8}{gkai}}
\AtEndDocument{\clearpage\end{CJK*}}");

import three;
size(200);
draw(unitbox,red);
label("顶点  $A$ ",(1,1,1),2Z);

```

1.6 如何编辑 asy 文件

通常情况下, 在命令行反复执行交互式的绘图是不方便的. 因此我们通常都是采用批处理形式, 即把代码保持为一个以 `.asy` 为扩展名的文件, 然后通过编辑器去调用 `asy` 命令来编译该文件. 我们需要配置一下我们手头的编辑器, 才能真正开始地利用 Asymptote 语言绘图.

1.6.1 用 emacs 编辑器

emacs 的用户可以把下面的内容 (主要内容是安装目录中的 `asy-init.el`) 放到 emacs 的 `.emacs` 文件当中,

```
;;(add-to-list 'load-path "C:/Program Files/Asymptote")
(add-to-list 'load-path "/usr/local/share/asymptote")
(autoload 'asy-mode "asy-mode.el" "Asymptote major mode." t)
(autoload 'lasy-mode "asy-mode.el" "hybrid Asymptote/Latex major mode." t)
(autoload 'asy-insinuate-latex "asy-mode.el" "Asymptote insinuate LaTeX." t)
(add-to-list 'auto-mode-alist '("\\.asy$" . asy-mode))
```

或者更简单一些的

```
;;(add-to-list 'load-path "C:/Program Files/Asymptote")
(add-to-list 'load-path "/usr/local/share/asymptote")
(require 'asy-mode)
```

Windows 的用户采用第一行. 用 C-c C-c 可以编译和预览当前缓存区域中的 `asy` 文件. 而且当光标落在某个函数上面时, 可以用 C-c ? 查看该函数的用法, 用 C-x ` 可以查错. 详细的功能介绍用 C-h f `asy-mode` <RET> 查看. 另外, 请注意那个 `asy-mode.el` 与 `asy-keywords.el` 是否都在 emacs 的装载目录中, 例如最好是让上述两个 .el 文件在同一目录下.

1.6.2 用 vim 编辑器

vim 的用户可以把 `/usr/local/share/asymptote` 目录下的 `asy.vim` 复制到 `~/.vim/syntax` 目录下. (Windows 下的 gVim 是 `C:\Program Files\Vim\vimfiles\syntax` 等目录). 然后把下面几行加入到 `~/.vimrc` 里面 (Windows 下的 gVim 在 `C:\Program Files\Vim` 目录下的 `_vimrc` 文件).

```
augroup filetypedetect
au BufNewFile,BufRead *.asy      setf asy
augroup END
filetype plugin on
```

如果发现还是没有语法高亮, 可以在 `~/.vimrc` 加上一句 `syntax on`.

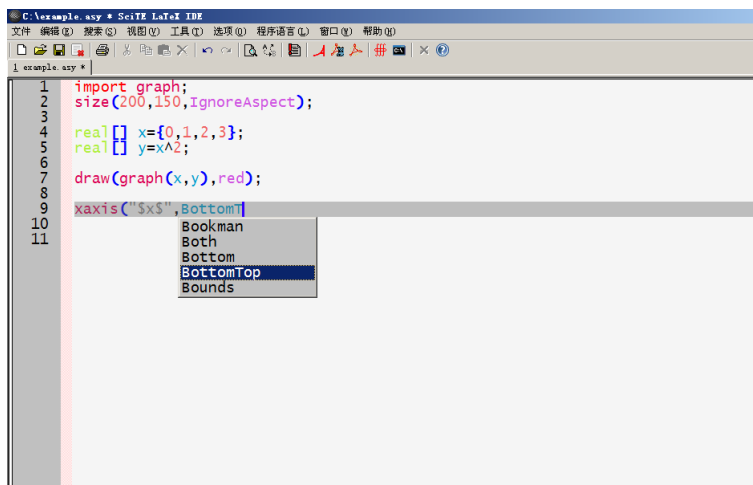
如果想用 `:make` 编译和预览当前文件, 可以把下面几行加入到 `~/.vim/ftplugin/asy.vim` 中 (Windows 下的 gVim 是 `C:\Program Files\Vim\vimfiles\ftplugin\asy.vim`).

```
setlocal makeprg=asy\ %
setlocal errorformat=%f:\ %l.%c:\ %m
```

对于其他编辑器, 如 SciTE, NotePad++, Crimson Editor, gedit 等等, 都有相应的语法高亮, 且它们的语法高亮规则基本上是取自 `asy-keyword.el`, 即分别为 `asy-keywork-name`, `asy-type-name`, `asy-function-name`, `asy-variable-name` (它们是 Asymptote 的保留关键词, 数据类型名, 函数名, 变量名) 采用不同的颜色.

1.6.3 用 SciTE 编辑器

要想在 SciTE 里面有 Asymptote 语法高亮和自动补全, 暂时还只能用 instanton 老师的改良版本 LaTeXIDE. 见 <http://code.google.com/p/scitelatexide/downloads/list> 的. 效果如下图.



其中那个 `asy.api` 几乎包括了 Asymptote 所有的关键词, 它可以利用 `asy-keywords.el` 生成的. 由于 `asy-keywords.el` 是随着版本更新而更新的, 因此, 如果我们希望用上最新的关键词, 可以进行如下操作来更新那个 `asy.api`.

```

file fin=input("asy-keywords.el");
file out=output("asy.api");
string[] one=fin.word();
int[] index;
int[] rindex;
for(int i=0;i<one.length;++i){
if(find(one[i],"(")!=-1) index.push(i);
if(find(one[i],")")!=-1) rindex.push(i);
}
string[] [] keywords;
for(int i=0;i<index.length;++i){
keywords[i]=one[index[i]+1 : rindex[i]];
}
string[] asykeywords=concat(...keywords);
asykeywords=sort(asykeywords);
for(int i=0;i<asykeywords.length;++i){
//write(out,asykeywords[i],suffix=endl);
write(out,asykeywords[i]+'\\r\\n');
}

```

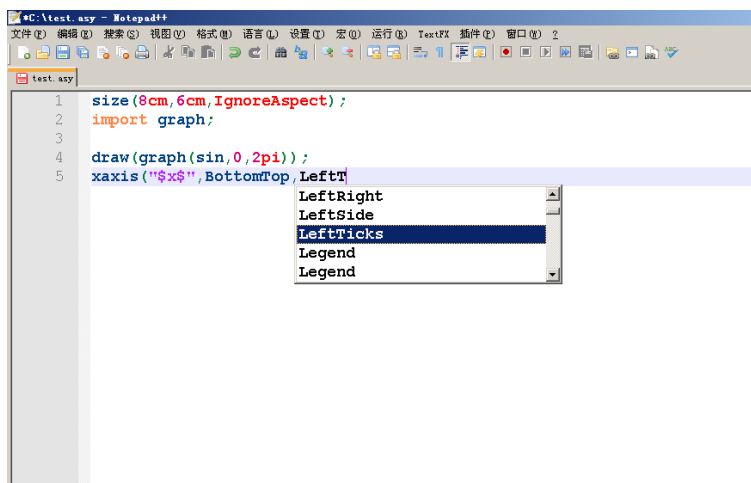
把上述代码保存为 `asyapi.asy` 文件, 让它与 `C:\Program Files\Asymptote` 目录下的 `asy-keywords.el` 放在一起, 然后执行 (或双击执行)

```
asy asyapi.asy
```

那么就会在当前目录生成 `asy.api` 这个文件, 把它复制到 LaTeXIDE 的 SciTE.exe 所在目录就可以了.

1.6.4 用 NotePad++ 编辑器

NotePad++ 语法高亮和自动补全关键词的效果如图所示.



ollydbg 做了一个关于本配置非常漂亮的视频, 已经打包在 <http://code.google.com/p/asy4cn/downloads/list> NotesOnAsy-*.tar.gz 的 ollydbg 目录中.

配置 NotePad++ 的 asy 语法高亮和自动补全

我们可以用 asy 自身来生成那个 `userDefineLang.xml` 文件, 方法如下. 在 <http://code.google.com/p/asy4cn/downloads/list> 的 NotesOnAsy-*.tar.gz 里面找到 `userDefineLangxml.asy` 这个文件, 让它与 `C:\Program Files\Asymptote` 目录中的 `asy-keywords.el` 放在同一目录下 (这个 `asy-keywords.el` 文件也可以在 `asymptote-*.src.tgz` 这个源代码包里面找到), 接着在命令行下执行 (或双击执行)

```
asy userDefineLangxml.asy
```

那么就会在当前目录生成 `userDefineLang.xml` 文件, 把它复制到 `C:\Documents and Settings\ 您的用户名\Application Data\Notepad++` 目录下. 如果一切正常, 那么重新打开 NotePad++, 这时应该在 语言 (Language) 的最下方出现 asy, 我们把它选上. 现在我们可以用 NotePad++ 打开一个以 `asy` 为扩展名的文件, 那么就会有语法高亮显示.(注意, 如果是采用绿色版本的 NotePad++, 那么直接把 `userDefineLang.xml` 放在解压后 `notepad++.exe` 所在目录就可以了)

类似的, 在 <http://code.google.com/p/asy4cn/downloads/list> 的 NotesOnAsy-*.tar.gz 里面找到 `asyxml.asy` 这个文件, 让它与 `C:\Program Files\Asymptote` 中的 `asy-keywords.el` 放在同一

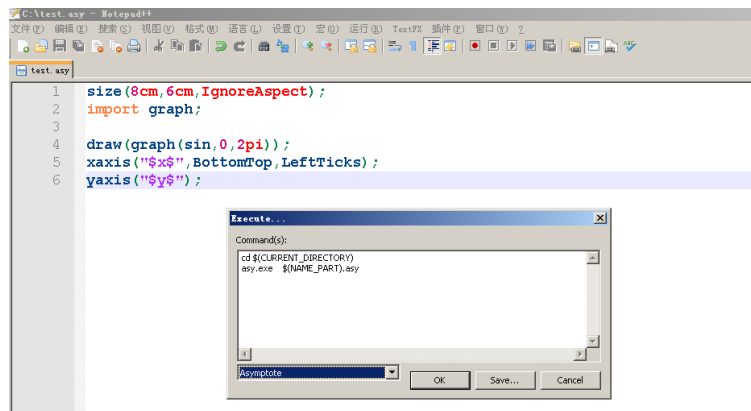
目录下 (这个 `asy-keywords.el` 文件也可以在 `asymptote-*.src.tgz` 这个源代码包里面找到), 再执行 (或双击执行)

```
asy asyxml.asy
```

同样, 会在当前目录生成 `asy.xml` 文件, 把它复制到 `C:\Program Files\Notepad++\plugins\APIs` 目录下 (或 NotePad++ 那个绿色版本的相应的目录)

接下去我们还要选择启用自动补全这个功能. 这时依次选择 设置 (Settings), 首选项 (Preferences), 备份与自动完成 (Backup / Auto-completion), 所有输入均启用自动完成 (Enable Auto-completion on each input) 中, 把 函数自动完成 (Function completion) 项选上. 重新打开 Notepad++, 我们可以试试输入一些代码, 这时将会有代码提示, 可以用 `Enter` 和 `Tab` 选择和补全.

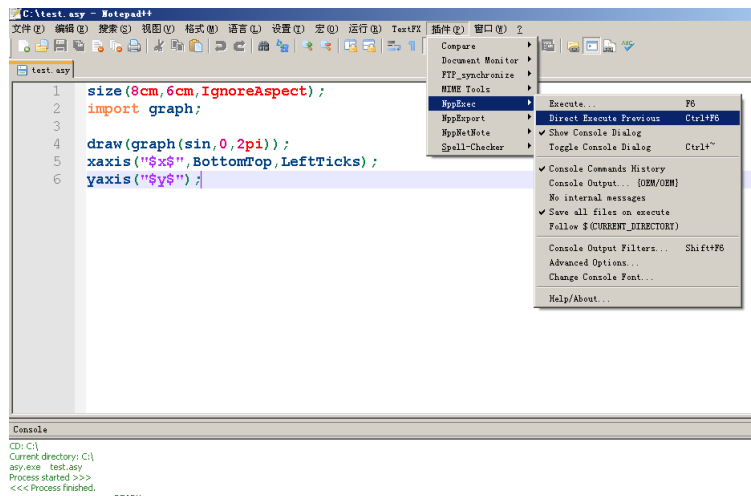
配置 NotePad++ 编译当前 asy 文件



编辑完 `asy` 后, 如何希望编译当前的 `asy` 文件, 那么可以选择 插件 (Plugins), `NppExec`, `Execute F6` 或直接用 `F6`, 然后在弹出窗口填上

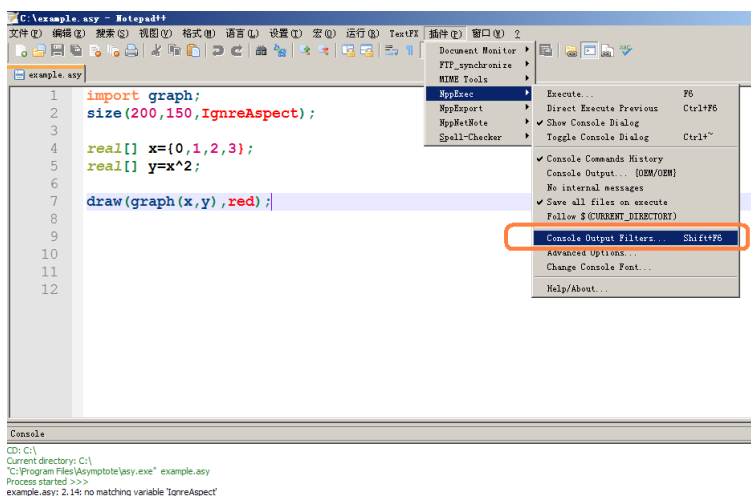
```
cd $(CURRENT_DIRECTORY)
```

```
asy.exe $(NAME_PART).asy
```

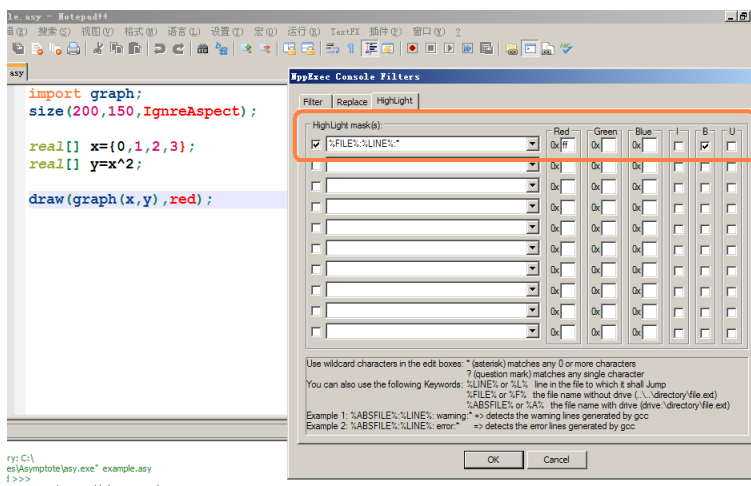


然后我们把 *Save all files on execute* 等勾上, 那么在编译之前就会自动保存当前文档. 注意, 我们还可以用 **Ctrl+F6** 重复执行上一次的编译命令. 不妨给刚才的编译命令起一个名字, 比如 **Asymptote**, 保存好下次还可以再调用. 另外, 那个 **F6** 和 **Ctrl+F6** 的这种快捷键的绑定是可以在 **设置 (Settings)**, **管理快捷键 (Short Mapper)**, **Plugin commands (Plugin commands)** 里面更改的.

配置 NotePad++ 定位语法出错行



我们依次点 **插件, NppExec, Console Output Filter**, 进入设置栏.



因为我们观察到 **asy** 的出错信息的格式是

文件名.**asy** : 行数. 列数 : 警告信息

因此, 我们在第一行填上

%FILE%.asy:%LINE%:*

依次代表带扩展名的文件名, 出错的行数, 警告信息. 然后选上颜色. (我们这里选了红色 **FF 00 00** 和粗体 **B**)

```

1 import graph;
2 size(200,150,IgnreAspect);
3
4 real[] x={0,1,2,3};
5 real[] y=x^2;
6
7 draw(graph(x,y),red);
8
9
10
11
12

```

```

Console
C:\>
Current directory: C:\
"C:\Program Files\Asymptote\asy.exe" example.asy
=====
example.asy: 2:14: no matching variable 'IgnreAspect'
<<< Process finished
===== READY =====

```

双击出错信息就会跳的源文件相应的那一行. 显然, 这个功能非常便于我们查错!

配置 NotePad++ 编译嵌入 asy 代码的 tex 文件

类似地, 我们可以编译一个嵌入了 asy 代码的 tex 文件, 当按下 F6 以后, 我们在输入框填上 (这里以 xelatex 为例)

```

cd $(CURRENT_DIRECTORY)
xelatex.exe $(NAME_PART).tex
asy.exe $(NAME_PART)-*.asy
xelatex.exe $(NAME_PART).tex
"C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe" $(NAME_PART).pdf

```

并且把它起一个名字, 确定以后就会依次用 xelatex, asy, xelatex 编译相应的文档, 最后用 Adobe Reader 打开生成的 pdf. 也可以采用 latexmk.pl 这个脚本, 具体用法参考第 185 页, 此时在输入框填上

```

cd $(CURRENT_DIRECTORY)
perl -S latexmk.pl -pdf $(NAME_PART).tex
"C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe" $(NAME_PART).pdf

```

如果是希望用 gsvie32.exe 预览 pdf, 那么把上面做法中的最后一行如下改动, 因为 gsvie32.exe 似乎比较特殊.

```
cmd /k "C:\Program Files\Ghostgum\gsview\gsview32.exe" $(NAME_PART).pdf &EXIT
```

1.7 卸载

对 Linux i386 可执行发行版本, 可以执行

```

tar -zxvf asymptote-x.xx.i386.tgz | xargs --replace=% rm /%
texhash

```


其中 `x.xx` 是版本号.

如果是从源代码编译的, 请进入该目录, 然后执行

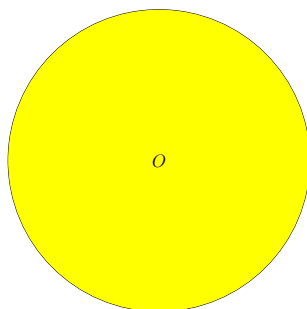
```
sudo make uninstall
```


第二章 2D 绘图

2.1 基本的绘图命令

Asymptote 的绘图基于 `draw`, `fill`, `clip` 以及 `label` 四个原始的命令. 前三个根源于 PostScript 的绘图命令, 最后一个用于在图形上添加字母和图形, 此处即调用了 \LaTeX 的功能.

举一个综合的例子.



```
size(200,200);  
draw(unitcircle);  
fill(unitcircle,yellow);  
label("$O$",(0,0));
```

每一个完整的命令语句以 ; 结尾.

其中 `size(200,200);` 可以理解为我们输出图形大小是 200×200 的图形. 可以省略为 `size(200);`.

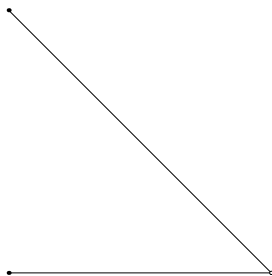
`unitcircle` 是 Asymptote 预定义的路径, 表示半径为 1 的圆. 现在调用 `draw` 命令画出.

同理, 其后的一句是在封闭的路径中填上黄色.

最后一句是在坐标为 $(0,0)$ 处标上字母 O , 其中引号内的 `O` 会被输送到一个 \TeX 文件中生成相应的字母, 然后再标在 $(0,0)$ 位置.

我们先试试一些简单的绘图例子, 等对 `asy` 语言有一定了解以后才开始学习它的语法.

2.2 画线段和点



```
size(200);
pair A=(0,0), B=(1,0), C=(0,1);
draw (A--B--C);
dot(A);
dot(B,UnFill);
dot(C);
```

`pair` 用来声明一种称为二元组的数据类型, 表示平面坐标. 现在对 A , B , C 同时声明, 用了逗号连接 (称为逗号运算符).

`--` 表示用直线连接两点.

`dot` 用于画出一个实心点, 其后加上 `UnFill` 就得到空心的点. 这个 `UnFill` 真正的把点挖空, 而不是用白色填充, 这是 Asymptote 提供的重要功能之一.

2.3 画虚线



```
size(200);
pair A=(0,0), B=(1,0), C=(1,1), D=(0,1);
draw(A--B,solid);
draw(B--C,dashed);
draw(C--D,dashdotted);
draw(D--A,dotted);
```

其中 `solid` 可以省略, 表示画实线;

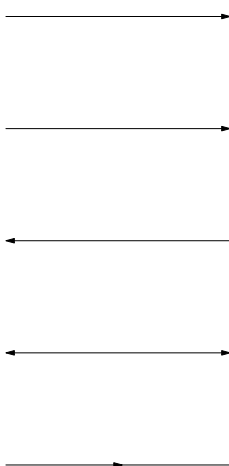
`dashed` 画虚线.

`dashdotted` 画点划线.

`dotted` 画实心点线.

更多线的类型将在后面阐述.

2.4 画箭头



```
size(200,0);
draw((0,3)--(2,3),Arrow);
draw((0,2)--(2,2),EndArrow);
draw((0,1)--(2,1),BeginArrow);
draw((0,0)--(2,0),Arrows);
draw((0,-1)--(2,-1),MidArrow);
```

`Arrow` 与 `EndArrow` 效果一样, 都是在路径的末端添加箭头.

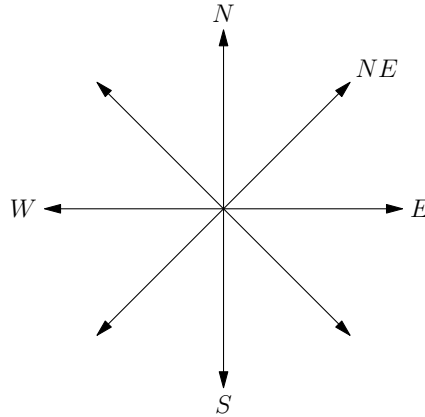
`BeginArrow` 在路径的开头加箭头.

`Arrows` 在路径的头尾都加上箭头.

`MidArrow` 则是在路径的中间添加箭头.

`Arrow` 其实是一个函数, 我们可以带上一些参数控制它的大小, 形状以及安放的位置, 这些在后面再详细论述.

Asymptote 预先定义了一些表示方向的二元组, 比如 `E=(1,0)`, `S=(0,-1)`, `W=(-1,0)`, `N=(0,1)` 以及它们的复合 `NE` 等等. 因为二元组同时也表示平面坐标系中的点, 因此, 这些方向同时也表示点. 举一个综合一点的例子,



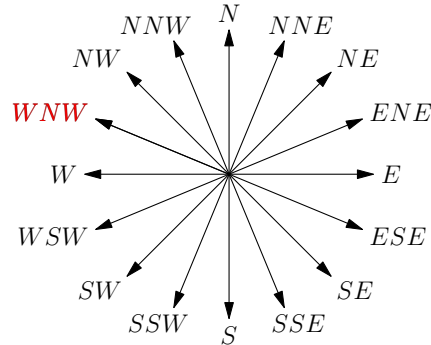
```
size(200);
pair O=(0,0);
draw(O--E,Arrow);
draw(O--NE,Arrow);
draw(O--N,Arrow);
draw(O--NW,Arrow);
draw(O--W,Arrow);
draw(O--SW,Arrow);
draw(O--S,Arrow);
draw(O--SE,Arrow);
label("$E$",E,E);
label("$N$",N,N);
label("$W$",W,W);
label("$S$",S,S);
label("$NE$",position=NE,align=NE);
```

其中 `label` 是 Asymptote 里面的一个函数, 用于标字母或图形, 可以事先指定标的位置 (`position`) 以及相对与那个位置的放法 (`align`), 因此, 该函数的调用格式之一是:

```
label(Label,position,align);
```

如最后一行我们把 `NE` 这个 `Label` 标在 `position=NE` 这个点的东北 (`align=NE`).

从刚才那个例子, 我们可以初步了解到 Asymptote 中的函数都可以加上一些数据, 我们把这些数据称为该函数的参数. 另外, 函数也可以复合. 比如我们可以让 `draw` 和 `Label` 函数复合, 以便同时画方向以及在其实标字母.

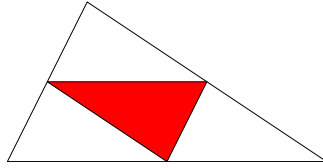


```
size(200);
pair O=(0,0);
draw(Label("$E$",EndPoint),O--E,Arrow);
draw(Label("$N$",EndPoint),O--N,Arrow);
draw(Label("$W$",EndPoint),O--W,Arrow);
draw(Label("$S$",EndPoint),O--S,Arrow);
draw(Label("$NE$",EndPoint),O--NE,Arrow);
draw(Label("$NW$",EndPoint),O--NW,Arrow);
draw(Label("$SE$",EndPoint),O--SE,Arrow);
draw(Label("$SW$",EndPoint),O--SW,Arrow);
draw(Label("$NNE$",EndPoint),O--NNE,Arrow);
draw(Label("$NNW$",EndPoint),O--NNW,Arrow);
draw(Label("$SSE$",EndPoint),O--SSE,Arrow);
draw(Label("$SSW$",EndPoint),O--SSW,Arrow);
draw(Label("$ENE$",EndPoint),O--ENE,Arrow);
draw(Label("$ESE$",EndPoint),O--ESE,Arrow);
draw(Label("$WNW$",EndPoint),O--WNW,Arrow);
draw(Label("$WSW$",EndPoint),O--WSW,Arrow);
draw(Label("$WNW$",position=EndPoint,red),O--WNW,Arrow);
```

当然,更为重要的时,我们可以同时控制 `Label` 与 `draw` 的关系,从而达到把 `Label` 标在我们希望的位置. 特别的,看最后一行, `position=EndPoint` 表明我们把 `Label` 标在路径 `O--WNW` 的末端.

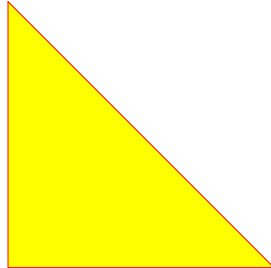
2.5 画线段按比例的分点

对两个点 A 和 B 构成的线段 (我们后面称为 `path` 的数据类型), 我们可以用 `midpoint` 函数来求它们的中点. 调用格式是: `midpoint(path)`



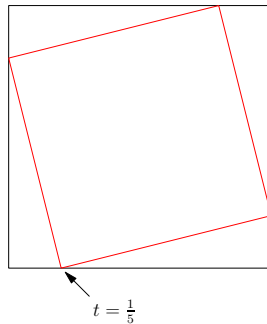
```
size(200);
pair A, B, C;
A=(0,0); B=(4,0); C=(1,2);
pair D,E,F;
D=midpoint(A--B);
E=midpoint(B--C);
F=midpoint(C--A);
draw(A--B--C--cycle);
filldraw(D--E--F--cycle,red);
```

其中 `--cycle` 表示我们用 `--` 这种连接方式使得所画的路径是封闭而且是周期循环的. 特别要注意的是: `A--B--C--cycle` 与 `A--B--C--A` 是不一样的. 后者并不形成一个封闭路径, 它只是一条首尾巧好是同一个点的路径. 因此, 也只有前者才可以在有它围成的封闭区域填上颜色. 那个 `filldraw` 就是把路径填上颜色的同时把该路径围成的封闭区域也填上颜色. 这两个颜色可以不一样.



```
size(200);
filldraw((0,0)--(0,2cm)--(2cm,0)--cycle,fillpen=yellow,drawpen=red);
```

一般的, 我们用 `T interp(T a, T b, real t)` 表示 $a + t \cdot (b - a)$, 即从 `a` 往 `b` 方向走到占整段比例为 `t` 的地方, 于是当比例 `t=1/2` 是就是中点. 另外, `T` 可以代任意有加减乘运算的一些诸如 `real`, `pair` 等的数据类型.



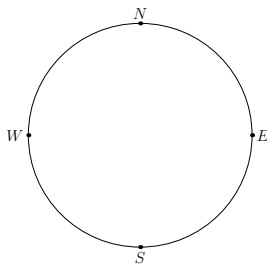

```

size(200);
pair A,B,C,D;
A=(0,0);B=(1,0);C=(1,1);D=(0,1);
real t=1/5;
pair X=interp(A,B,t);
pair Y=interp(B,C,t);
pair Z=interp(C,D,t);
pair W=interp(D,A,t);
draw(A--B--C--D--cycle);
draw(X--Y--Z--W--cycle,red);
arrow("$t=\frac{1}{5}$",X,SE);

```

2.6 画过若干个点的曲线

用 `..` 表示这种与直线不一样的连接两个点方式. 这种画曲线原理将在后面讲到. 现在我们可以利用四个点 E , N , W , S 来画出一个与圆非常接近的图形.

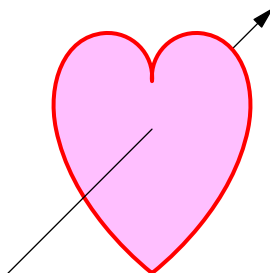


```

size(200);
draw(E..N..W..S..cycle);
dot("$E$",E,E);
dot("$N$",N,N);
dot("$W$",W,W);
dot("$S$",S,S);

```

下面我们利用曲线功能画一些有趣一点的图形.



```

size(200);
draw((0,0)--(2.5,2.5),linewidth(1),Arrow);
filldraw((0,1)..(1,2)..(2,0)..(1,-2)..(0,-3)
        ..(0,-3)..(-1,-2)..(-2,0)..(-1,2)..(0,1)..cycle,
        pink,red+linewidth(3));
draw((-3,-3)--(0,0),linewidth(1));

```

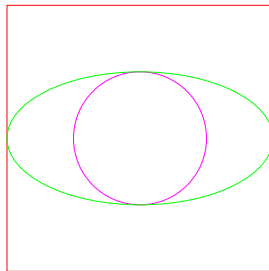
2.7 变量与函数

我们在前面其实已经接触到变量和函数这两个概念, 在这里阐述一下. Asymptote 中的变量就是在使用之前要预先定义好的数据类型. 比如前面的表示方向的 **E**, **N**, **W**, **S** 以及各种颜色 **red**, **green** 等等. 用户也可以自定义一些变量, 但一定要事先声明它的数据类型. 这种语法与 C++/Java 是类似的 (但本书不假定读者学过任何语言). 至于 函数, 在 Asymptote 中也把它看成变量, 我们说它是一种一阶变量, 于是普通的 变量 就可以看成零阶变量, 两个函数复合就是更高阶的变量. 这是 Asymptote 语言最为独特的一点, 也使得我们在函数中调用或定义函数非常方便. Asymptote 预先定义了很多画基本图形的函数, 可以直接调用, 不过我们心里要清楚函数需要的参数是什么. 这些常用的函数包括 (后面会看到, 如果调用

box(矩形的左下角, 矩形的右上角);

ellipse(椭圆的中心, 水平方向的轴长, 竖直方向的轴长);

drawline(直线的第一个点, 直线上的第二个点);

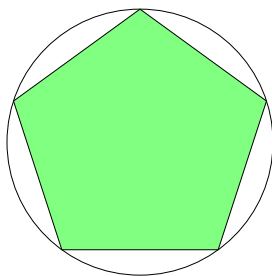


```

import graph;
size(200);
pair min=(-2,-2);
pair max=(2,2);
pair O=(0,0);
pair A=(2,0);
draw(box(min,max),red);
draw(circle(0,1),magenta);
draw(ellipse(0,2,1),green);

```

2.8 画正多边形

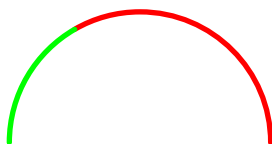


```
size(200);
draw(unitcircle);
filldraw(polygon(5),lightgreen);
```

2.9 画圆弧

圆弧是圆的一部分, 用 `arc` 函数.

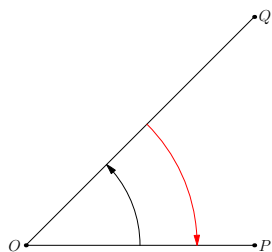
```
path arc(pair c, real r, real angle1, real angle2);
```



```
size(200);
defaultpen linewidth(4);
pair O=(0,0);
draw(arc(O,2cm,0,120),red);
draw(arc(O,2cm,120,180),green);
```

其中 `defaultpen linewidth(4);` 把默认的画笔的设为 4bp 粗细.

有时候我们希望画一个角之间的圆弧, 比如通常的平面几何的图形中给角做一个标记, 默认的作图没有这个功能, 这时就要调用 Asymptote 里面的 `geometry` 宏包.



```

import geometry;
size(200);
pair O=(0,0);
pair P=(2,0);
pair Q=(2,2);
dot("$P$",P);
dot("$Q$",Q);
dot("$O$",O,W);
draw(P--O--Q);
draw(arc(P,O,Q,1),Arrow);
draw(arc(Q,O,P,1.5),red,Arrow);

```

2.10 画直角记号

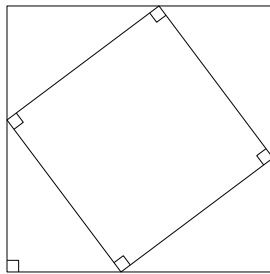
直角符号用 `perpendicular` 函数, 同样要调用 `geometry` 宏包. 基本的用法是

```

perpendicular(pair z,pair align,pair dir);
perpendicular(pair z,pair align,path g);

```

该直角是相对 $z \text{--} z + \text{dir}$ 的, `align` 表明它的安置的方位, 给出路径 `g` 的话, Asymptote 会把它转成该路径 `g` 的切线方向 (关于路径的切线方向我们以后阐述).



```

import geometry;
size(200);
real a=3,b=4,c=5;
pair A=(0,0);
pair B=(a+b,0);
pair C=(a+b,a+b);
pair D=(0,a+b);
pair X=(a,0);
pair Y=(a+b,a);
pair Z=(b,a+b);

```

```

pair W=(0,b);
draw(A--B--C--D--cycle);
draw(X--Y--Z--W--cycle);
perpendicular(A,NE,A--B);
perpendicular(X,NE,X--Y);
perpendicular(Y,NE,Y--Z);
perpendicular(Z,NE,Z--W);
perpendicular(W,NE,W--X);

```

2.11 自定义函数

我们也可以自己定义新函数. 假定我们要画一条抛物线 $y = x^2$, 我们要预先定义一个函数. 格式如下

```

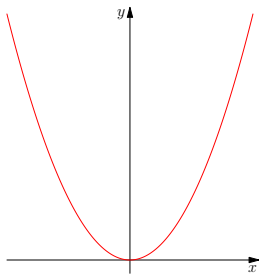
real f(real x){return y=x^2;};

```

其中 `real x` 声明函数自变量 `x` 是实数型, `f` 前面的 `real` 声明函数 `f` 也是一个实数型.

2.12 画函数图像

利用若干个点点画曲线能力有限, 我们可以利用数学中的函数图像来生成更丰富多彩而且有意义的曲线, 更进一步的, 我们可以生成参数曲线以及用隐函数表达的曲线.



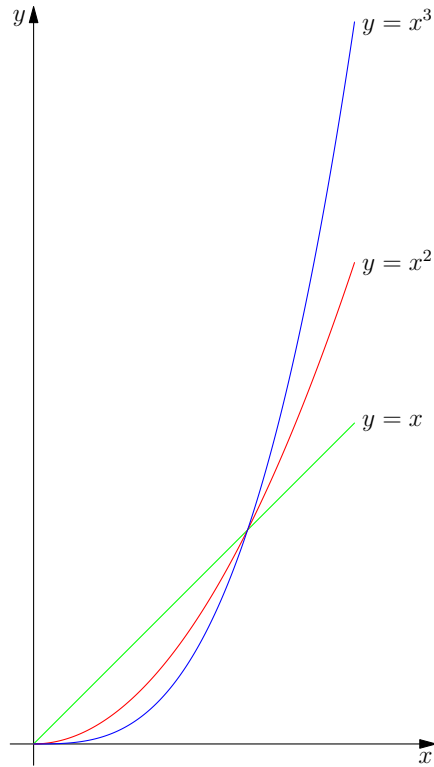
```

import graph;
size(200);
real f(real x){return x^2;};
path p=graph(f,-2,2,operator..);
draw(p,red);
xaxis("$x$",Arrow);
yaxis("$y$",Arrow);

```

函数 `graph` 是一个返回数据类型为路径 `path` (其实是称为 `guide` 的数据类型, 与 `path` 稍微有些区别) 的函数. 它需要知道要画的函数 `real f(real x)`, 所画函数自变量的变换范围, 还有就是用折线画还是用光滑曲线画. 因此, 前面代码调用它的格式如下:

```
guide graph(real f(real), real a, real b, interpolate join=operator --);
```



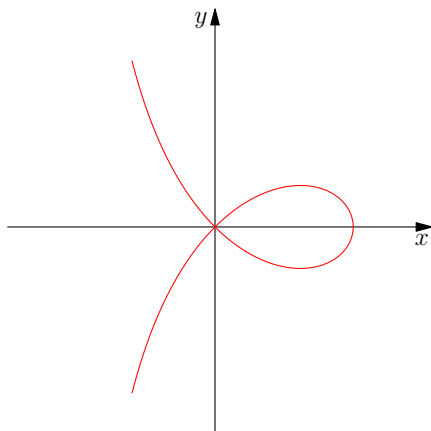
```
import graph;
size(200,0);
real f1(real x){return x;}
real f2(real x){return x^2;}
real f3(real x){return x^3;}
guide f1=graph(f1,0,1.5,operator..);
guide f2=graph(f2,0,1.5,operator..);
guide f3=graph(f3,0,1.5,operator..);
draw(f1,green);
draw(f2,red);
draw(f3,blue);
label(Label("$y=x$",position=EndPoint,align=E),f1);
label(Label("$y=x^2$",EndPoint,E),f2);
label(Label("$y=x^3$",EndPoint,E),f3);
xaxis("$x$",Arrow);
yaxis("$y$",Arrow);
```

在 `Label()` 函数中加 `position=EndPoint`, 那么就会标在所画曲线的末端, 也可以用完全等价的 `position=Relative(1)`.

类似地, 标 y 轴时用 `align=N` 把 y 安置 (`align`) 在 y 轴默认的 `position=EndPoint` 的北边.

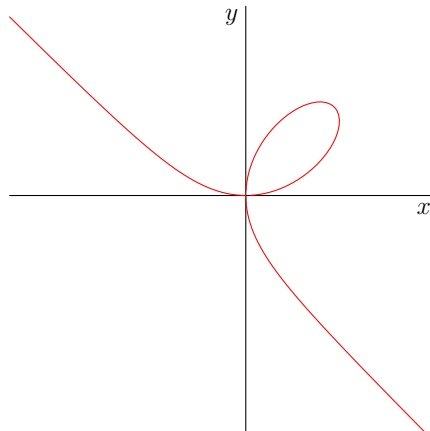
2.13 参数曲线和隐函数表达的曲线

参数方程的例子如下, 可以有多种方式, 比如采用注释号 `/* */` 内部的方案.



```
import graph;
size(200);
pair f(real t){
    real x=(1-t^2)/(1+t^2);
    real y=t*(1-t^2)/(1+t^2);
    return (x,y);
}
draw(graph(f,-2,2),red);
/*
    real x(real t) {return (1-t^2)/(1+t^2);}
    real y(real t) {return t*(1-t^2)/(1+t^2);}
    draw(graph(x,y,-2,2),red);
*/
limits((-1.5,-1.5),(1.5,1.5));
xaxis("$x$",Arrow);
yaxis("$y$",Arrow);
```

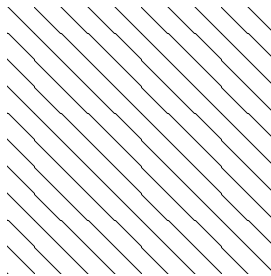
隐函数表达的曲线要调用 `contour` 宏包. 我们要画的是 Descartes 的叶形线, 方程为 $x^3 + y^3 - 3axy = 0$, 也就是二元函数 $x^3 + y^3 - 3axy$ 在 0 处的等值线, 因此我们要采用匿名函数新建一个数组, 该数组仅仅包含一个元素 {0}, 即用 `new real[] {0}` 的方式.



```
import graph;
import contour;
size(200);
real a=1;
real f(pair t){
    real x=t.x;
    real y=t.y;
    return x^3+y^3-3a*x*y;
}
guide[] g=contour(f,(-4,-4),(4,4),new real[] {0},100);
draw(g,red);
axes("$x$","$y$");
```

2.14 画阴影

传统的图形, 特别是印成黑白颜色的图形, 通常是希望在封闭曲线里面填上阴影而不是颜色. Asymptote 的宏包 `patterns` 提供了一些模式, 我们先讲最常见的一种 `hatch` 画阴影.



```
size(200);
import patterns;
```



```
add("name",hatch(NW));
fill(unitsquare,pattern("name"));
```

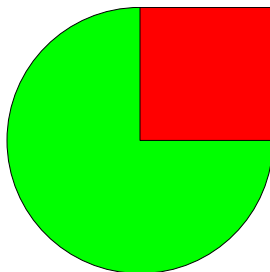
`hatch(NW)` 是一种西北走向的阴影斜线的图形. 用 `add("name",hatch(NW));` 命名为"name".

接下去用 `pattern("name")` 的方式把它做成一个类似与颜色的画笔.

`hatch()` 函数还有其他参数, 比如线的粗细, 线的间隔等.

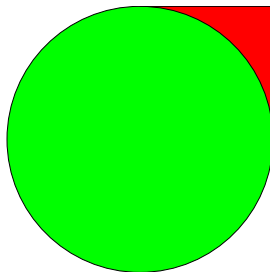
2.15 子图

在 Asymptote 中我们可以独立的画一些图, 然后再分别添加到图上. 这时, 我们可以定义一些称为 `picture` 的数据类型.



```
size(200);
picture pic;
picture fig;
filldraw(pic,unitcircle,green);
filldraw(fig,unitsquare,red);
add(pic);
add(fig);
```

这与直接画的效果似乎是一样的, 不过用这种独立画的方式的其中一个便利的地方是可以随意控制图片的画出顺序. 这时候用 `above=true` 和 `above=false` 控制的.



```
size(200);
picture pic;
```

```

picture fig;
filldraw(pic,unitcircle,green);
filldraw(fig,unitsquare,red);
add(pic);
add(fig,above=false);

```

我们看到, 现在最后加上的图反而给前面画的图盖住了. 这种灵活的独立图形的输出的是独立画图的好处之一. 这个也用在下面的 `clip` 命令用法当中以及以后的动画制作.

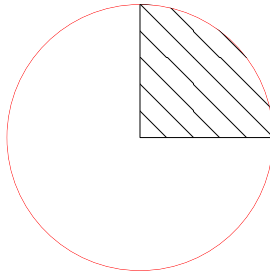
2.16 裁剪命令 clip

除了前面已经阐述的 `draw`, `fill`, `label` 三个基本命令以外, 还有一个 `clip` 命令, 可以借助于封闭曲线裁剪出该曲线围成的区域里面的部分.

```

clip(picture pic=currentpicture, path g, stroke=false, pen fillrule=currentpen);

```



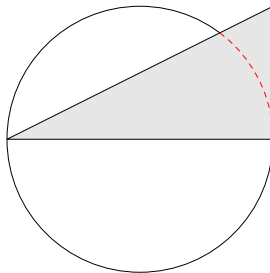
```

size(200);
import patterns;
draw(unitcircle,red);
add("name",hatch(NW));
filldraw(unitsquare,pattern("name"));

clip(currentpicture,unitcircle);

```

利用 `clip` 这个函数, 我们可以任意地裁剪出我们所想要的东西, 然后再用 `add` 函数添加.



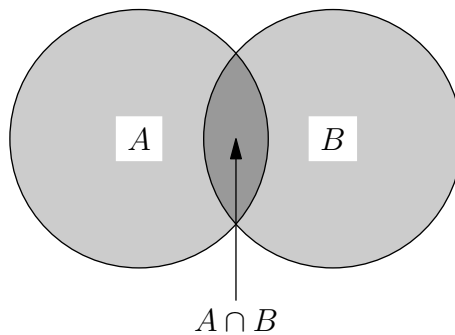
```

size(200);
draw(unitcircle);
path p=(-1,0)--(1,1)--(1,0)--cycle;
filldraw(p,lightgray);
picture pic;
draw(pic,unitcircle,dashed+red);
clip(pic,p);
add(pic);

```

我们定义了一个子图 `pic` 用来独立画一个虚线单位圆, 然后用路径 `p` 把它剪下来. 最后再用 `add` 添加到图上.

下面的 Venn 图的例子运用上面技巧画出了两个集合交集部分的.



```

size(7cm,0);
pair C1=(-3,0),C2=(3,0);
real r=4;
path circleA=circle(C1,r);
path circleB=circle(C2,r);

fill(circleA,0.8white);
fill(circleB,0.8white);

picture pic;
fill(pic,circleA,0.6white);
clip(pic,circleB);
add(pic);

draw(circleA);
label("$A$",C1,UnFill(1mm));

```

```
draw(circleB);
label("$B$",C2,UnFill(1mm));
draw(Label("$A\cap B$",0),(0,-5)--(0,0),Arrow);

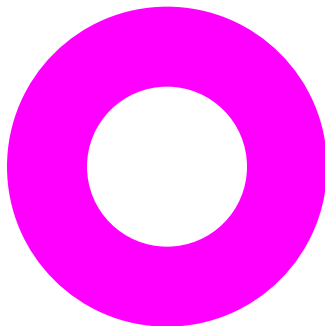
shipout(bbox(5mm,invisible));
```

2.17 unfill 命令

Asymptote 利用 `clip` 这个功能定义了一个非常好用的 `unfill` 命令, 可以利用封闭曲线来挖洞. 前面我们已经见过 `UnFill`, 再举一个例子.

2.18 奇偶法则填颜色

我们前面已经看到 `fill` 命令可以在封闭循环的路径内填上各种颜色.



```
size(200);
pair O=(0,0);
path p1=circle(0,1);
path p2=circle(0,2);
fill(p1^^p2,evenodd+magenta);
```

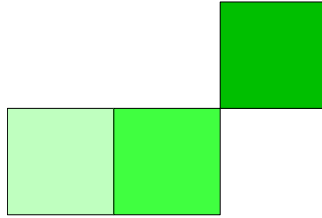
2.19 变换

变换可以使得我们可以利用原来的图形来生成新的图形. Asymptote 可以进行一般的仿射变换, 即可以进行旋转, 伸缩, 旋转, 反射等变换以及它们的复合.

平移变换是

```
transform shift(real x,real y);
transform shift(pair z);
```

分别为依照向量 (x,y) 和 z 平移.



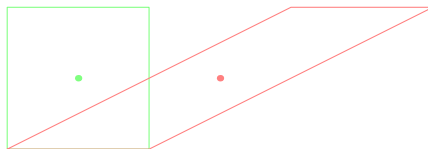
```
size(200);
filldraw(unitsquare,palegreen);
filldraw(shift(1,0)*unitsquare,mediumgreen);
filldraw(shift(2,1)*unitsquare,heavygreen);
```

伸缩变换包括

```
transform xscale(real x);
transform yscale(real y);
transform scale(real s);
transform scale(real x, real y);
```

```
transform slant(real s);
```

错切变换 `slant` 变换把点 (x,y) 变换为 $(x+s*y,y)$.

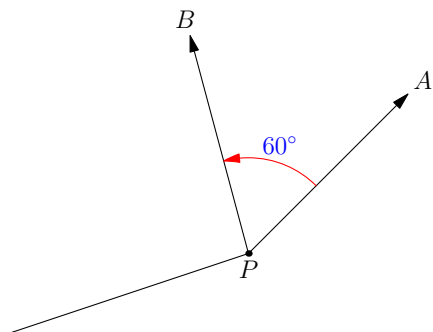


```
size(200);
draw(unitsquare,lightgreen);
draw(slant(2)*unitsquare,lightred);
pair C=(0.5,0.5);
dot(C,lightgreen);
dot(slant(2)*C,lightred);
```

从图中可以看到, 点 $(0.5, 0.5)$ 经过错切 `slant(2)` 变换后变成点 $(1.5, 0.5)$. 经过简单的运算, 我们可以知道 $s = \tan \alpha$, 其中 α 是错切变换 `slant` 的倾斜角.

```
transform rotate(real angle, pair z=(0,0));
```

旋转变换 `rotate` 是绕着点 z 旋转 `angle` 度数. 默认是绕着坐标原点旋转.



```
import geometry;
size(200);
pair O=(0,0);
pair P=(3,1);
pair V=(2,2);
pair A=P+V;
pair B=rotate(60,P)*A;
draw(O--P);
dot("$P$",P,S);
draw(Label("$A$",EndPoint),P--A,Arrow);
draw(Label("$B$",EndPoint),P--B,Arrow);
draw(Label("$60^\circ$",blue),arc(A,P,B,1.2),red,Arrow);
```

```
transform reflect(pair a, pair b);
```

反射变换是以 $a--b$ 这条直线为对称轴反射.

2.20 简单的编程

Asymptote 的语法类似于 C, C++, 以及 Java. 不过本书不假定读者有任何的编程语言背景, 反而是希望直接学习 asy 这种有趣的绘图语言.

我们在前面已经接触了它的一些语法, 这里归纳一下. 首先, 每一个变量都要声明它的数据类型. 比如

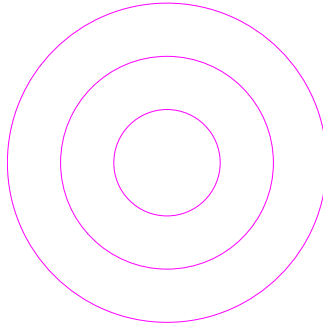
```
real x;
pair z;
```

然后对应地赋予相应的值.

```
x=1.0;
z=(3.0,4.0);
```

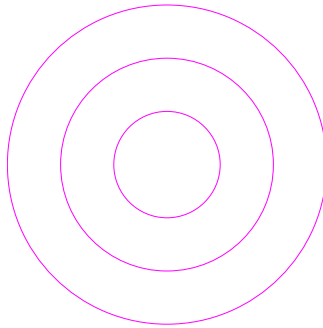
最后, 就是一些程序设计语言都有的判断语句和循环语句. 我们在这里说一下用的最多的 `for` 循环语句.

假定我们要画半径分别为 1, 2, 3,... 的几个同心圆, 我们当然可以如下画出:



```
defaultpen(magenta);  
size(200);  
pair O=(0,0);  
draw(circle(O,1));  
draw(circle(O,2));  
draw(circle(O,3));
```

这也未尝不可, 可是再要求多画几个, 大家也就会觉得烦了. 编程语言都提供一种叫 `for` 循环的语句去干这个事情.



```
defaultpen(magenta);  
size(200);  
pair O=(0,0);  
for(int i=1;i<4;++i){  
    draw(circle(O,i));  
}
```

我们解释一下 `for` 括号里面三个语句的意思.

第一个语句 `int i=1;` 是对变量 `i` 赋予初始值 1;

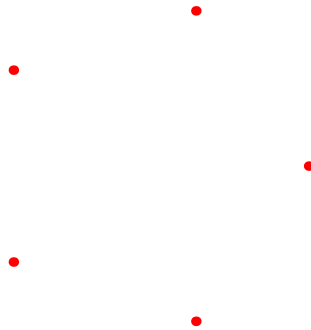
第二个语句 `i<4`; 是判断 `i` 是否小于 4, 如果是, 就执行 `draw(circle(0,i));`

第三个语句 `++i` 把 `i` 增加 1.

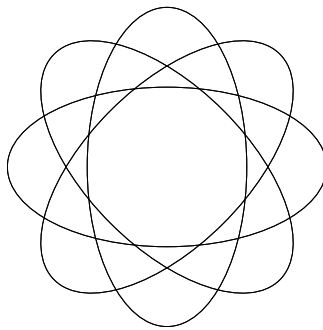
按照这个规律, 与 `i=1` 对应的 `draw(circle(0,1))` 是最先执行的. 然后 `i++` 使得 `i` 从 1 增加到 2, 仍然满足 `i<4` 这个的条件, 因此, 接下去可以执行 `i=2` 对应的 `draw(circle(0,2))`. 做完以后把 `i++` 又使得 `i` 从 2 增加到 3, `i=3` 对应的 `draw(circle(0,3))` 被执行. 最后, 变量 `i` 增加到 4. 由于此时 `i` 已经增加到了 4, 不再满足 `i<4` 的条件, 因此 `for` 就此停止工作.

我们看到, 现在也是恰好画了半径分别为 `i=1, i=2, i=3` 的三个同心圆, 是不是很神奇!

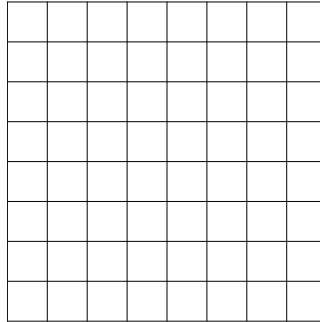
下面再列举一些简单的例子.



```
size(200);
int n=5;
for(int i=0;i<n;++i){
    dot(dir(i*360/n),red+6pt);
}
```

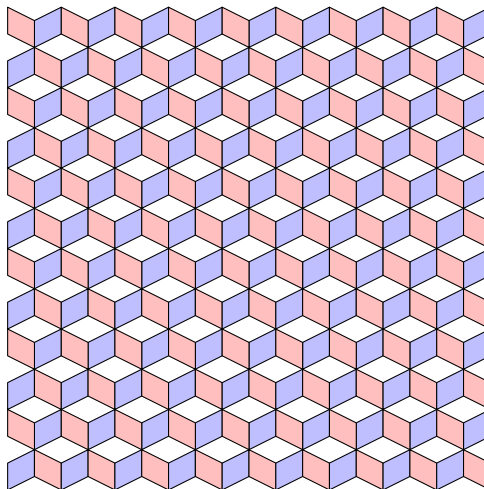


```
size(200);
int n=8;
real step=360/n;
for (int i=0; i < n; ++i) {
    draw(rotate(i*step)*xscale(.5)*unitcircle);
}
```

```
size(200);
for(int i=0;i<=8;++i){
    draw((i,0)--(i,8));
    draw((0,i)--(8,i));
}
```

下面的例子稍微复杂一些, 即先在一个 `picture pic;` 里面画好一个基本的图形, 然后通过平移达到铺砌平面.



```
size(8cm,0);

path A=(0,0)--(2,1)--(2,3)--(0,2)--cycle;
path B=reflect((0,0),(0,1))*A;
path C=rotate(180,(0,0))*A;
path D=reflect((0,0),(1,0))*A;

picture pic;

filldraw(pic,A^^C,lightblue+white,black);
```

```

filldraw(pic,B^^D,lightred+white,black);

for(int i=0; i<9; ++i){
    for(int j=0; j<6; ++j){
        add(shift(4*i,6*j)*pic);
    }
}

```

由于循环的应用大部分是与数组结合起来的, 因此我们后面还有讨论一下数组以及它的应用, 到时候我们有更为精彩的例子可以介绍.

2.21 与向量 pair 有关的一些函数

我们先归纳一下与向量 pair 这种数据类型有关的一些知识.

int 整数类型; 如果没有给定初始式, 默认值为 0. 整数允许的最小值为 `intMin`, 最大值为 `intMax`. 我们可以用

```

write(intMin);
write(intMax);

```

看看这两个数.

real 实数; 它将设为计算机结构的本地浮点类型的最大精度. 实数的隐式初始式为 0.0. 实数具有精度 `realEpsilon`, 有效数字为 `realDigits`. 最小的正实数为 `realMin`, 而最大的正实数为 `realMax`.

pair 向量, 也看成是复数, 实数构成的有序对 (x, y) . 复数 z 的实部和虚部可读为 $z.x$ 和 $z.y$. 我们称 x 和 y 为复数数据元素的虚拟成员; 然而, 它们不能直接修改. 复数的隐式初始式为 $(0.0, 0.0)$.

```
pair conj(pair z)
```

返回 z 的复共轭.

```
real length(pair z)
```

返回向量 z 的长度, 即复数的模.

```
real abs(pair z)
```

与 `length(pair)` 功能一样.

```
real angle(pair z)
```

返回 z 的幅角, 单位为弧度, 在区间 $[-\pi, \pi]$ 内取值.

```
real degrees(pair z, bool warn=true)
```

返回 z 的幅角, 单位为度, 在区间 $[0, 360)$ 内, 当 `warn` 为 `false` 且 $z.x=z.y=0$ 时返回 0 (而不是产生错误). 这个 `warn=false` 可以在与极坐标互相转换时避开 $(0,0)$ 这个奇点.

```
pair unit(pair z)
```

返回与复数 z 同方向的单位向量.

```
pair expi(real angle)
```

返回以弧度角 `angle` 为方向的单位向量, 这里用 `expi` 这个符号是源于 $e^{i\theta}$.

```
pair dir(real degrees)
```

返回以角度 `degrees` 为方向的单位向量,

```
real xpart(pair z)
```

返回 $z.x$.

```
real ypart(pair z)
```

返回 $z.y$.

```
real dot(pair z, pair w)
```

返回点积 (内积) $z.x*w.x+z.y*w.y$

```
pair minbound(pair z, pair w)
```

返回 $(\min(z.x, w.x), \min(z.y, w.y))$.

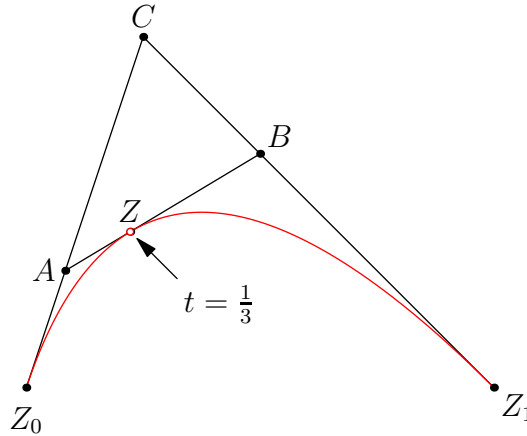
```
pair maxbound(pair z, pair w)
```

返回 $(\max(z.x, w.x), \max(z.y, w.y))$.

2.22 Bezier 曲线的原理及与 path 有关函数

Asymptote 画曲线与通常的其他软件不一样, 并非采用折线逼近来生成曲线的办法, 而是采用一种称为 Bezier 曲线办法. 我们在此花一些篇幅来介绍这个概念. 因为一旦我们理解 Asymptote 这种生成曲线或曲面的方法, 将会有助于我们理解 Asymptote 的与路径 `path` 相关的一些命令, 做到心中有数, 也有助于灵活地解决问题.

我们前面已经懂得如何画直线段, 如何生成曲线呢? 1959 年, 法国人 Paul de Casteljau(与此独立地, 1962 年的 Bezier) 发现了现在称为 Bezier 曲线的生成光滑曲线的办法. 我们在此阐述一下其中巧妙的想法.



```
import graph;
size(200);
pair Z0=(0,0),Z1=(4,0),C=(1,3);
real t=1/3;
pair A=interp(Z0,C,t);
pair B=interp(C,Z1,t);
pair Z=interp(A,B,t);
draw(Z0--C--Z1);
draw(A--B);
arrow("$t=\frac{1}{3}$",Z,SE);
pair f(real t){return (1-t)^2*Z0+2t*(1-t)*C+t^2*Z1;}
draw(graph(f,0,1),red);
dot("$A$",A,W);
dot("$B$",B,NE);
dot("$C$",C,N);
dot("$Z_{0}$",Z0,S+S);
dot("$Z_{1}$",Z1,SE);
dot("$Z$",Z,N);
dot(Z,red,UnFill);
```

给出 Z_0 和 Z_1 两个点, 我们只能得到连接它们的直线段, 此时如果给出一个时刻 t , 那么我们可以通过这个时刻访问到这条线段上以 t 为比例的点, 即 $(1-t)Z_0 + tZ_1$. de Casteljau 天才的想法是添加一个额外的点 C , 并且把它称为控制点. 然后我们顺次连接 Z_0, C, Z_1 , 只能得到一段折线, 下面我们按照一个比例 t (我们也称为时刻 t), 比如 $t = \frac{1}{3}$, 然后在 Z_0 到 C 的 $\frac{1}{3}$ 处取一个点 A , 同样, 在 C 到 Z_1 的 $\frac{1}{3}$ 处取另外一个点 B , 连接 A 与 B , 然后在 A 到 B 的 $\frac{1}{3}$ 处再取点 Z , 那么就称 Z 为对应于这个时刻 $\frac{1}{3}$ 一个点. 我们可以想象, 当那些时刻遍历 0 到 1 的所有时刻, 就得到一条轨迹, 通过简单的代数

运算, 我们可以依次得到

$$A = (1 - t)Z_0 + tC \quad (2.1)$$

$$B = (1 - t)C + tZ_1 \quad (2.2)$$

$$Z = (1 - t)A + tB \quad (2.3)$$

把 (2.1) 和 (2.2) 代入 (2.3), 我们可以得到 Z 与 t 的关系.

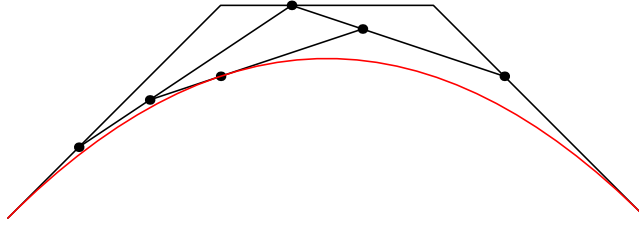
$$Z(t) = (1 - t)^2 Z_0 + 2t(1 - t)C + t^2 Z_1$$

这是一条过 Z_0 和 Z_1 的二次参数曲线.

我们上面得到的是 2 次的参数曲线, 如果采用两个控制点, 那么我们就能够得到 3 次的 Bezier 曲线, 其中的原理是类似的, 请观察图.

$$(1 - t)^3 z_0 + 3t(1 - t)^2 c_0 + 3t^2(1 - t)c_1 + t^3 z_1$$

Asymptote 里面就是采用这种 3 次曲线. 我们可以把这种生成曲线的方式推广到曲面.



```
size(200);
pair z0=(0,0),c0=(1,1),c1=(2,1), z1=(3,0);
draw(z0--c0--c1--z1);
real t=1/3;
pair m0=interp(z0,c0,t);
pair m1=interp(c0,c1,t);
pair m2=interp(c1,z1,t);
pair m3=interp(m0,m1,t);
pair m4=interp(m1,m2,t);
pair m5=interp(m3,m4,t);
dot(m0);
dot(m1);
dot(m2);
dot(m3);
dot(m4);
dot(m5);
draw(m0--m1--m2);
```

```
draw(m3--m4);
draw(z0..controls c0 and c1..z1,red);
```

下面我们介绍一下与路径 path 有关的一些函数.

```
pair precontrol(path p, int t)
```

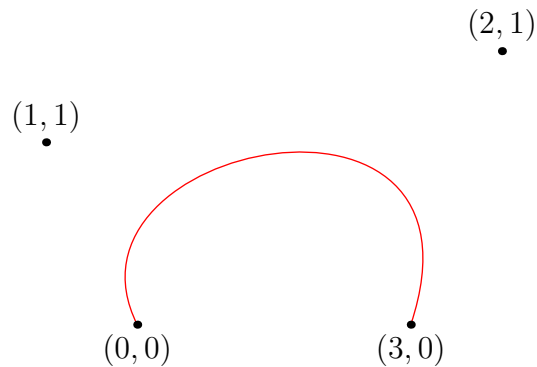
```
pair postcontrol(path p, int t)
```

```
int length(path p)
```

一般的曲线是由若干条 Bezier 曲线连接起来的, 这里返回的是路径 p 的总的分段数目. 比如我们可以用 `write(unitcircle); write(length(unitcircle));` 看看 `unitcircle` 的构造以及它的 `length`.

```
pair point(path p, real t)
```

我们在前面已经介绍了 Bezier 曲线的生成原理, 从中我们可以知道, 对于一条这种两个控制点的 3 次 Bezier 曲线, 随便给出一个时刻 t , 我们是简单的得出时刻 t 对应的点.



```
size(200);
pair z0=(0,0),c0=(-1,2),c1=(4,3), z1=(3,0);
path p=z0..controls c0 and c1..z1;
draw(p,red);
dot("$ (0,0) $",z0,S);
dot("$ (1,1) $",c0,N);
dot("$ (2,1) $",c1,N);
dot("$ (3,0) $",z1,S);

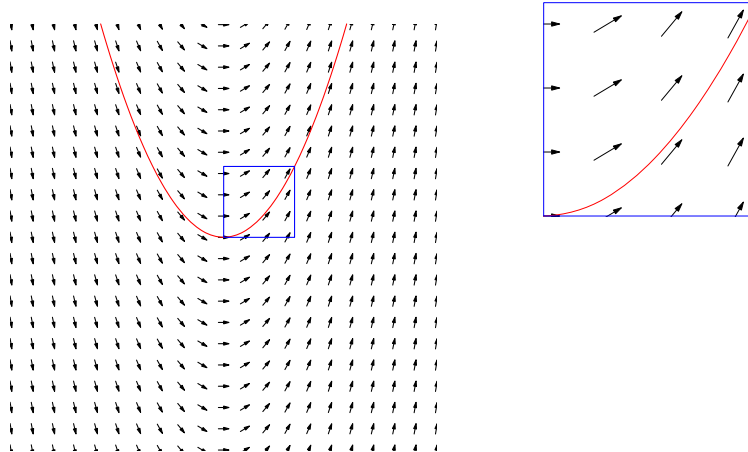
write(precontrol(p,0));
write(postcontrol(p,0));

write(precontrol(p,1));
write(postcontrol(p,1));
```

2.23 数组

我们经常要表示一系列对象 A_0, A_1, A_2, \dots , 它们具有相同的类型.

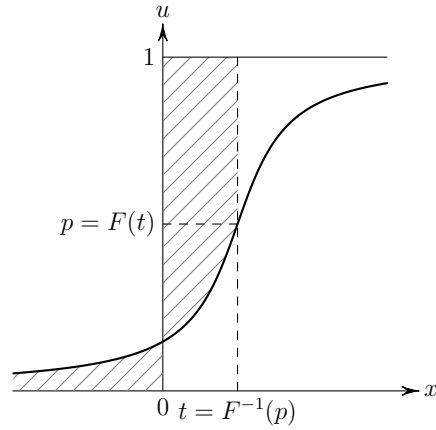
2.24 subfigure



```
import slopefield;
picture base;
size(base,200);
real func(real x) {return 2x;}
add(base,slopefield(func,(-3,-3),(3,3),20,Arrow));
draw(base,curve((0,0),func,(-3,-3),(3,3)),red);
draw(base,box((0,0),(1,1)),blue);
add(base.fit());

clip(base,box((0,0),(1,1)));
add(scale(0.5)*base.fit(),(150,10));
```

2.25 graph 宏包



```

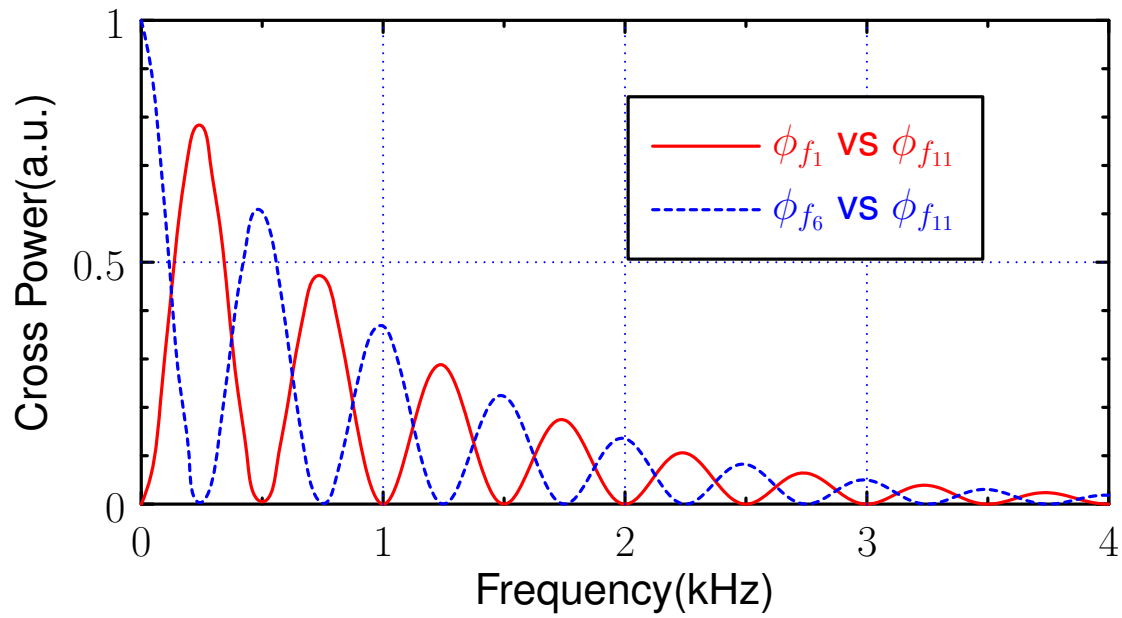
size(200,IgnoreAspect);
import graph;
import patterns;

add("hatch",hatch(H=2mm,dir=NE,gray));
real f(real t ) {return 1/2+atan(t-2)/pi;}
real a=-4;
real b=2;
real c=6;
path p=(a,0)--(0,0)--(0,1)--(b,1)--(b,f(b))..(graph(f,b,a))..(a,f(a))--cycle;
fill(p,pattern("hatch"));

draw(graph(f,a,c,operator ..),linewidth(1bp));
draw((0,1)--(c,1));
draw((b,0)--(b,1),dashed);
draw((b,f(b))--(0,f(b)),dashed);
label("$t=F^{-1}(p)$",(b,0),S);
label("$p=F(t)$",(0,f(b)),W);
label("$0$", (0,0),S);
label("$1$", (0,1),W);

axis(Label("$x$",align=2E),Arrow(HookHead,5));
axis(Label("$u$",align=2N),0,1.1,Arrow(HookHead,5));

```

```

import graph;
unitsize(4cm,8cm);
defaultpen(1.5pt+fontsize(20pt)+Helvetica());
real f(real x){return sin(2pi*x)^2*exp(-x);}
real g(real x){return cos(2pi*x)^2*exp(-x);}

draw(graph(f,0,4,operator..),red,Label("$\phi_{f_1}$ vs $\phi_{f_{11}}$"));
draw(graph(g,0,4,operator..),blue+linetype("2 2"),Label("$\phi_{f_6}$ vs $\phi_{f_{11}}$"));

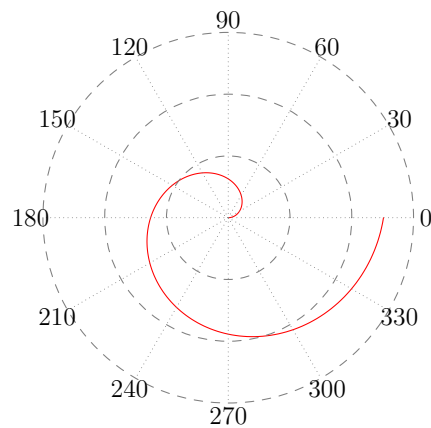
xaxis("Frequency(kHz)",BottomTop,LeftTicks);
yaxis("Cross Power(a.u.)",LeftRight,RightTicks());

xequals(1,blue+Dotted);
xequals(2,blue+Dotted);
xequals(3,blue+Dotted);
yequals(0.5,blue+Dotted);

add(legend(),point((0)),NE,UnFill);

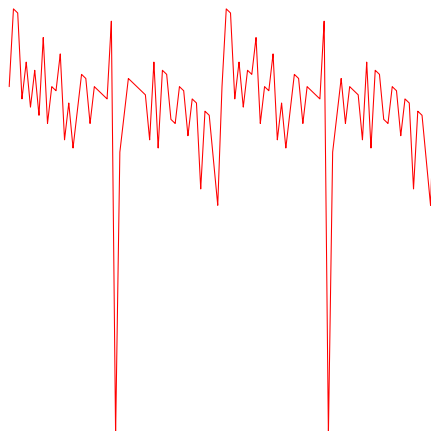
```

2.26 极坐标



```
size(200);
import graph;
pair O=(0,0);
real f(real r){return 2*r;}
draw(polargraph(f,0,2pi,operator..),red);
for(int i=0;i<4;++i){
    draw(circle(O,5*i),grey+dashed);
}
int n=12;
for(int i=0;i<n;++i){
    label(string(i*30),16*dir(i*30));
    draw(O--15*dir(i*30),gray+dotted);
}
```

2.27 数项级数表示的函数



```

import graph;
size(200,IgnoreAspect);

typedef real func(real);

func f(int n){
    return new real (real x){return (n*x-1/2-floor(n*x-1/2));};
    //real g(real x){return (n*x-1/2-floor(n*x-1/2));}; return g;
}

int n=35;
real Sum(real x){
    real s=0;
    for(int i=0;i<n;++i){
        s=s+f(i)(x);
    }
    return s;
}

draw(graph(Sum,0,2,operator--),red);

```

首先, 我们要定义一个函数族 $f_n(x)$, 为此我们定义与 n 相关的函数 $f(n)$, 由于函数是一种实型返回实型的数据类型, 这个不是已有的类型, 因此我们需要用 `typedef` 定义这种新的数据类型, 我们给它一个 `func` 的名称.

接下去用匿名函数 `new` 新建一个实型返回实型的函数, 然后返回函数 $f(n)$ 的表达式. 过程有些曲折, 不过这是因为 Asymptote 把函数也当成变量来处理的缘故.



```

import graph;
size(0,100);
real u(real x){return abs(x-floor(x+1/2));}

typedef real func(real);

func f(int n){

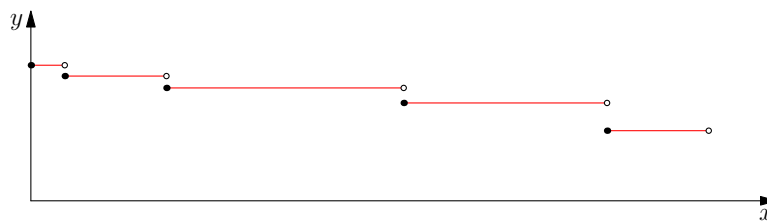
```

```

    return new real (real x) {return u(4^n*x)/4^n;};
}
int n=20;
real Sum(real x){
    real s=0;
    for(int i=0;i<n;++i){
        s=s+f(i)(x);
    }
    return s;
}
draw(graph(Sum,-1,1,operator--),red);

```

2.28 分段函数



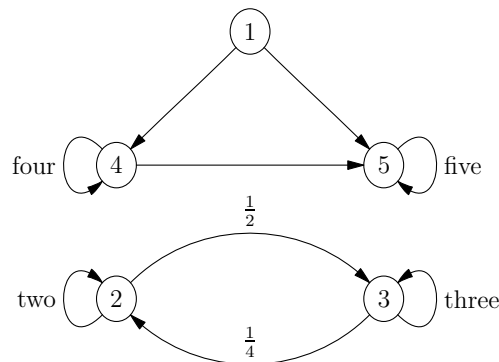
```

import graph;
size(0,100);

pair[] z={{(0,1/4),(1/4,1),(1,2+3/4),(2+3/4,4+1/4),(4+1/4,5)}};
real[] Y={1,0.92,0.8325,0.7217,0.5171};
for(int i=0;i<Y.length;++i){
    draw((z[i].x,Y[i])--(z[i].y,Y[i]),red);
    dot((z[i].x,Y[i]));
    dot((z[i].y,Y[i]),UnFill);
}
xaxis("$x$",0,5.5,Arrow);
yaxis("$y$",0,1.4,Arrow);

```

2.29 画线程图



```
size(200);
real margin=1mm;
pair A=(0,2), B=(-2,0), C=(2,0), D=(-2,-2), F=(2,-2);
object one=draw("$1$",ellipse,A,margin);
object four=draw("$4$",ellipse,B,margin);
object five=draw("$5$",ellipse,C,margin);
object two=draw("$2$",ellipse,D,margin);
object three=draw("$3$",ellipse,F,margin);

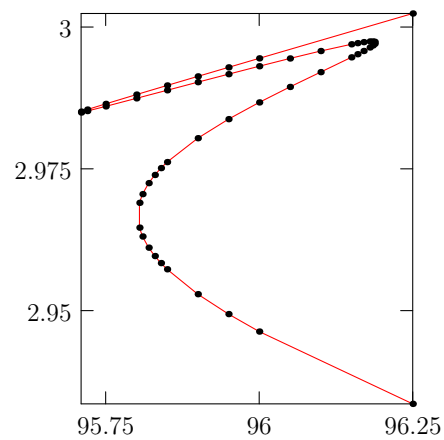
add(new void(picture pic, transform t) {
    draw(pic,Label("two",align=W),point(two,SW,t){SW}..{SE}point(two,NW,t),Arrow);
    draw(pic,Label("three",align=E),point(three,SE,t){SE}..{SW}point(three,NE,t),Arrow);
    draw(pic,Label("four",align=W),point(four,NW,t){NW}..{NE}point(four,SW,t),Arrow);
    draw(pic,Label("five",align=E),point(five,NE,t){NE}..{NW}point(five,SE,t),Arrow);
    draw(pic,point(one,SW,t){SW}..{SW}point(four,NE,t),Arrow);
    draw(pic,point(one,SE,t){SE}..{SE}point(five,NW,t),Arrow);
    draw(pic,point(four,E,t){E}..{E}point(five,W,t),Arrow);
    draw(pic,Label("$\frac{1}{2}$",align=N),
        point(two,NE,t){NE}..{SE}point(three,NW,t),Arrow);
    draw(pic,Label("$\frac{1}{4}$",align=N),
        point(three,SW,t){SW}..{NW}point(two,SE,t),Arrow);
});
```

2.30 读入数据绘图

假定我们有一些点的坐标, 我们有两种方法读入这些数据后绘图.

| | |
|--------|---------|
| 96.25 | 3.00239 |
| 96.00 | 2.99448 |
| 95.95 | 2.99289 |
| 95.90 | 2.9913 |
| 95.85 | 2.9897 |
| 95.80 | 2.98809 |
| 95.75 | 2.98645 |
| 95.72 | 2.98545 |
| 95.71 | 2.98509 |
| 95.71 | 2.98493 |
| 95.72 | 2.98519 |
| 95.75 | 2.98601 |
| 95.80 | 2.98743 |
| 95.85 | 2.98885 |
| 95.90 | 2.99028 |
| 95.95 | 2.99169 |
| 96.00 | 2.99309 |
| 96.05 | 2.99446 |
| 96.10 | 2.99577 |
| 96.15 | 2.99696 |
| 96.16 | 2.99716 |
| 96.17 | 2.99734 |
| 96.18 | 2.99746 |
| 96.185 | 2.99746 |
| 96.188 | 2.99739 |
| 96.188 | 2.99706 |
| 96.185 | 2.99676 |
| 96.18 | 2.9964 |
| 96.17 | 2.99578 |
| 96.16 | 2.99521 |
| 96.15 | 2.99466 |
| 96.10 | 2.99205 |
| 96.05 | 2.98944 |
| 96.00 | 2.98671 |
| 95.95 | 2.98376 |
| 95.90 | 2.9804 |
| 95.85 | 2.97619 |
| 95.84 | 2.97513 |

| | |
|--------|---------|
| 95.83 | 2.97392 |
| 95.82 | 2.97248 |
| 95.81 | 2.97054 |
| 95.805 | 2.96901 |
| 95.805 | 2.96463 |
| 95.81 | 2.96307 |
| 95.82 | 2.9611 |
| 95.83 | 2.95962 |
| 95.84 | 2.95838 |
| 95.85 | 2.95729 |
| 95.90 | 2.9529 |
| 95.95 | 2.94939 |
| 96.00 | 2.94631 |
| 96.25 | 2.93356 |



```

import graph;
size(200,IgnoreAspect);
file f=input("data.txt");
pair[] a;
while(true) {
    real x=f;
    real y=f;
    if(eof(f)) break;
    a.push((x,y));
}
draw(graph(a),red);
dot(a);

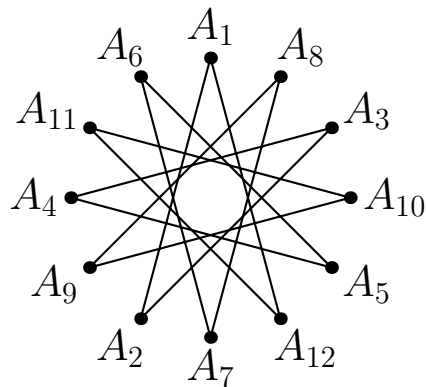
```

```
axis(BottomTop, LeftTicks);
axis(LeftRight, RightTicks);
```

另外一种做法如下.

```
import graph;
size(200, IgnoreAspect);
file filename=input("data.txt").line();
write(filename);
real[] [] A=filename.dimension(0,0);
A=transpose(A);
real[] x=A[0];
real[] y=A[1];
guide g = graph(x,y);
draw(g,red);
pair[] a=pairs(x,y);
dot(a);
axis(BottomTop, LeftTicks);
axis(LeftRight, RightTicks);
```

2.31 各种循环语句



```
size(200);
int n=12;
int k=5;
defaultpen(linewidth(1)+fontsize(20));
pair P(int i){return dir(90+k*(360/n)*i);}
Label L(int i){return Label(format("$A_{%d}$",i+1),align=P(i));}
pair[] star=sequence(P,n);
```



```
Label[] starlabel=sequence(L,n);
dot(starlabel,star);
draw(operator--(...star)--cycle);
```

2.32 弯曲路径上标标签

中华人民共和国万岁

```
texpreable("\usepackage{CJKutf8}
\AtBeginDocument{\begin{CJK*}{UTF8}{gbsn}}
\AtEndDocument{\clearpage\end{CJK*}}");
size(200);
import labelpath;
path p=reverse(rotate(-90)*yscale(2)*xscale(4)*unitcircle);
labelpath("中华人民共和国万岁",p,red);
//draw(p);
```

中华人民共和国万岁

```
texpreable("\usepackage{CJKutf8}
\AtBeginDocument{\begin{CJK*}{UTF8}{gbsn}}
\AtEndDocument{\clearpage\end{CJK*}}");
size(200);
path p=reverse(arc((0,0),1,20,160));
string[] fonts={"中","华","人","民","共","和","国","万","岁"};
pen[] colors={cyan,magenta,yellow,green,blue,red};
colors.cyclic=true;
for(int i=0;i<fonts.length;++i){
    label(scale(2)*fonts[i],point(p,i/(fonts.length-1)*length(p)),colors[i]);
}
draw(p,lightgray);
```



```

tex preamble("\usepackage{CJKutf8}
\AtBeginDocument{\begin{CJK}{UTF8}{gbsn}}
\AtEndDocument{\clearpage\end{CJK}}");

size(200);
path p=reverse(arc((0,0),1,20,160));
string[] fonts={"中","华","人","民","共","和","国","万","岁"};
pen[] colors={cyan,magenta,yellow,green,blue,red};
colors.cyclic=true;
for(int i=0;i<fonts.length;++i){
    real a=i/(fonts.length-1)*length(p);
    pair Position=point(p,a);
    pair T=dir(p,a);
    real Angle=degrees(T);
    label(rotate(Angle,Position)*scale(2)*fonts[i],Position,colors[i]);
}
draw(p,lightgray);

```

2.33 随机



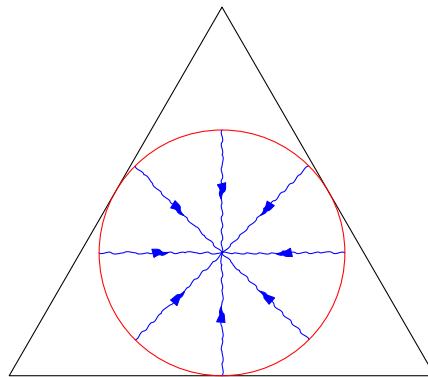
```

size(200);
pen colorpen=blue;
pair end=(0,0);
pair direction=E;
srand(seconds());
void randomwalk(int n){
    if (n==0)return;
    else{
        real angle=rand()%360;
        real lambda=1;
        pair begin=end;
        end=begin+scale(lambda)*direction;
        path randompath=begin--end;
        draw(randompath,colorpen);
        direction=rotate(angle)*direction;
        randomwalk(n-1);
    }
}
randomwalk(1000);

```

2.34 模拟徒手绘图的效果

Ivaldi 提供的 trembling 宏包可以模拟徒手绘图, 比较有趣.



```

import trembling;
size(200);
pair[] compass={E,S,W,N,NE,NW,SE,SW};
for(pair z: compass){
    path p=tremble(addnodes(z..(0,0)),angle=20,frequency=0.1,random=100);
}

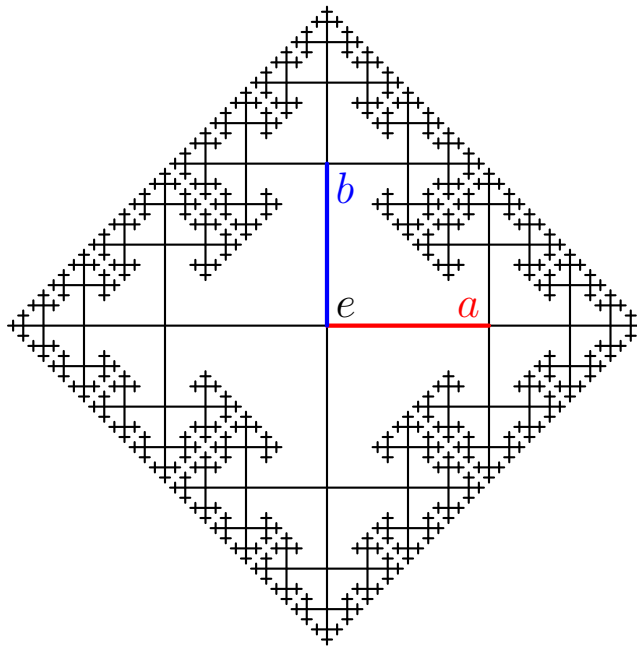
```

```

    draw(p,blue,MidArrow);
}
startTrembling(angle=10,frequency=0.1,random=80);
triangle T=triangleabc(2*sqrt(3),2*sqrt(3),2*sqrt(3));
draw(shift(-sqrt(3),-1)*T);
draw(unitcircle,red);

```

2.35 分形图



```

size(300,0);
defaultpen(linewidth(1pt)+fontsize(20));
real shortening=0.5;
void tree(pair A,pair B,int n){
    pair C,D,M;
    C=B+shortening*(rotate(90,B)*A-B);
    D=B+shortening*(rotate(-90,B)*A-B);
    M=B+shortening*(rotate(180,B)*A-B);
    if(n>0){
        draw(A--B);
        tree(B,C,n-1);
        tree(B,D,n-1);
        tree(B,M,n-1);
    }
}

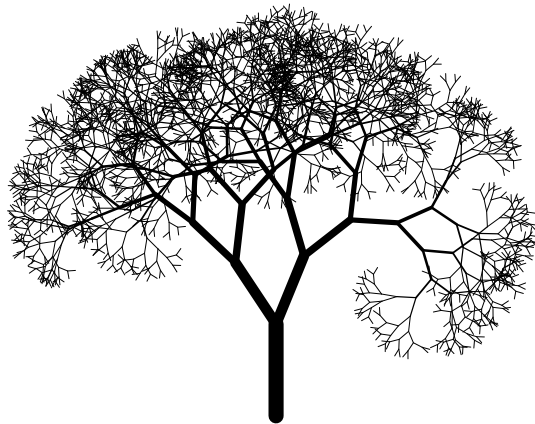
```

```

}
else{
    draw(A--B);
    draw(B--C);
    draw(B--D);
    draw(B--M);
}
}
tree((0,0),(0,1),4);
tree((0,0),(1,0),4);
tree((0,0),(-1,0),4);
tree((0,0),(0,-1),4);
draw(Label("$a$",EndPoint,NW),(0,0)--(1,0),red+2pt);
draw(Label("$b$",EndPoint,SE),(0,0)--(0,1),blue+2pt);
label("$e$",(0,0),NE);

```

下面画树的代码是从 *Learning METAPOST by Doing* 的 METAPOST 代码翻译过来的.



```

size(200);
real branchrotation=60;
real offset=180-branchrotation;
real thinning=0.7;
real shortening=0.8;
srand(seconds());
void tree(pair A,pair B,int n,real size){
    pair C,D;
    real thickness=size;
    real r_angle=offset+(branchrotation*(unitrand()));
    C=B+shortening*(rotate(r_angle,B)*A-B);

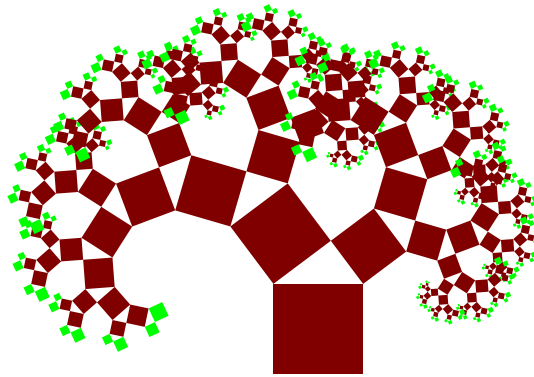
```

```

real l_angle=offset+(branchrotation*(unitrand()));
D=B+shortening*(rotate(-l_angle,B)*A-B);
if(n>0){
    draw(A--B,linewidth(thickness));
    thickness=thinning*thickness;
    tree(B,C,n-1,thickness);
    tree(B,D,n-1,thickness);
}
else{
    draw(A--B,linewidth(thickness));
    thickness=thinning*thickness;
    draw(B--C,linewidth(thickness));
    draw(B--D,linewidth(thickness));
}
}
tree((0,0),(0,1),10,2mm);

```

我们也可以把 examples 目录下的 PythagoreanTree.asy 改编一下, 只画出一层绿色的叶子.



```

size(200);
real a=3;
real b=4;
real c=hypot(a,b);
void Tree(pair A,pair B,int n){
    pair C=rotate(-90,B)*A;
    pair D=rotate(90,A)*B;
    pair K=interp(D,rotate(aCos(b/c),D)*C,b/c);
    if(n==0)
        fill(A--B--C--D--cycle,green);
    else{

```

```
    Tree(D,K,n-1);  
    Tree(K,C,n-1);  
    fill(A--B--C--D--cycle,brown);  
  }  
}  
Tree((0,0),(1,0),8);
```


第三章 3D 绘图

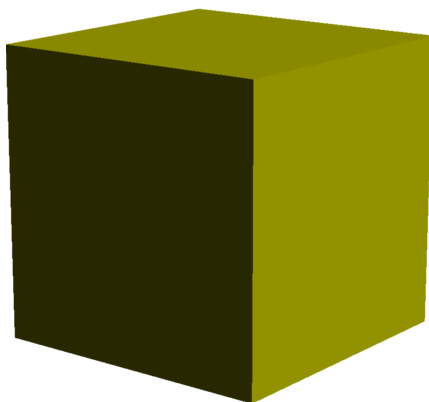
3.1 Asymptote 三维作图概述

与另外一种与 $\text{T}_{\text{E}}\text{X}$ 结合紧密的 `metapost` 语言不同, `Asymptote` 提供了一种原生的方式去生成三维的曲线, 主要是把 `MetaPost` 作者 `Hobby` 关于二维情形自动选择控制点生成曲线的方法推广到三维.

`Asymptote` 中与 3D 有关的主要是 `three`, `graph3`, `solids` 三个宏包, 其他辅助的宏包还有 `tube`, `grid3`, `contour3`, `obj`, `labelpath3` 等等. 在画 3D 图形之前, 我们一般都要调用前面三个宏包之一二.

3.2 预览 3D 输出

先给出一个简单的 3D 例子.



```
import three;  
size(200);  
draw(unitcube,yellow);
```

我们可以有四种预览这个 3D 的办法.

1. 使用 `Asymptote` 默认的基于 `OpenGL` 的预览. 即在命令行中使用

```
asy -V filename.asy
```

这时我们可以用鼠标左键控制弹出的窗口, 而在鼠标右键有各种选项. 下面只是说说比较常用的快捷键.

h: home(复位)

f: toggle fitscreen(全屏)

x: spin about the X axis(观察者绕着 X 轴旋转, 这里的 X 轴是指我们屏幕的水平方向)

y: spin about the Y axis(观察者绕着 Y 轴旋转, 这里的 Y 轴是指我们屏幕的竖直方向)

z: spin about the Z axis(观察者绕着 Z 轴旋转, 这里的 Z 轴是垂直屏幕指向我们的方向)

s: stop spinning(停止自旋)

m: rendering mode(solid/mesh/patch)(渲染的模式, 包括实物, 网格, 补丁等模式)

e: export(导出为 eps 或 pdf 格式)

c: show camera parameters(当使用此快捷键时, 将会在命令行输出 camera, target, up 等与投影有关的参数, 非常有利于我们根据这些参数调整最终的图形输出视角!)¹

+: expand(放大)

==: expand(放大)

-: shrink(缩小)

__: shrink(缩小)

q: exit(退出)

Ctrl-q: exit(退出)

当用 e 快捷键导出为 eps 或 pdf 时, 建议手动把弹出窗口最大化.

2. 输出为嵌入 PRC 真 3D 的 pdf. 这时采用

```
asy -f pdf filename.asy
```

我们的 TeX 系统需要有最新的 movie15 宏包, asy 会调用那个宏包把 asy 生成的 prc 文件嵌入到 pdf 中, 而且需要 Adobe 9.0 以上的版本的阅读器才能读出那个真 3D 效果.²

另外, 在 2010 年 6 月份, Asymptote 生成包含 PRC 的 pdf 文件的渲染速度有了极大的提高, 必要时候我们还可以加入

```
defaultrender=render(compression=Zero,merge=true);
```

¹具体说来, 就是当弹出 OpenGL 的窗口时, 随便用鼠标转动图像到合适位置, 然后用 c 键, 那么就会在命令行出现一大串关于视角的代码, 把它复制到代码里面, 然后再重新编译, 那么出来的图形的视角就是您所要的.

²对于 Linux 用户, 如果显示 3D 的画面不太稳定, 可以通过选择 Edit, 3D&Multimedia, Preferred Renderer, 把默认的 OpenGL(Mesa)(OpenGL 在 Linux 中的实现) 改为 Software.

`compression=Zero` 表示有损压缩参数为 0, 得到的 PRC 文件体积大一些, 但打开后很快就看到真 3D 效果, 且图形显示质量比较高. `merge=true` 表示渲染前隐没那些节点, 也使得我们转动真 3D 时有较高的显示速度, 更多的选项请看 `three.asy` 文件的注释. 必要时候我们可以把 `three.asy` 默认值改成我们常用的 (默认 `compression=High`, `merge=false`).

3. 可以不弹出预览窗口生成 eps 或 pdf, 用下面的命令 (以导出为 pdf 为例):

```
asy -noV -noprc -f pdf filename.asy
```

其中 `-noprc` 选项是指不把 prc 嵌入到 pdf 中, 取而代之是直接用那个基于 OpenGL 的 `render` (渲染) 把图像拍摄出来, 然后再生成 pdf. 如果导出的 `filename.pdf` 出现图形错乱的异常现象, 请尝试执行

```
asy -noV -f pdf -noprc -maxtile="(512,512)" filename.pdf
```

来生成 pdf 文件, 或者把下面代码临时加入到 `~/.src/config.asy` 里面或文件开头.

```
import settings;
glOptions="-indirect"; // 显卡驱动的选项, 一般也不用设.
prc=false; // 不采用嵌入 PRC 真 3D 格式.
maxtile=(512,512); // 控制弹出的 render 窗口的尺寸, 请作适当调整.
outformat="pdf"; // 输出为 pdf 格式.
interactiveView=false; // 交互式命令行时不预览
batchView=false; // 编译 asy 文件时不预览, 相当于 -noV.
```

另外, 在类 Unix 系统中用 asy 的上述 `-noV` 方式时, 在 `~/.asy/config.asy` 加入 `iconify=true`; 的选项可以使得不容易出现图像错乱的情形.

这种利用 OpenGL 生成的 3D 图像能做到自动消隐, 完全不用我们手工去使得被挡住的面不可见, 本文档的所有 3D 的图像就是这样生成的.

4. 当然, 我们可以生成 1.44 版本之前的 2D 向量图形式的假 3D, 主要缺点是在图像消隐方面目前还比较弱, 此时用 `asy -render=0 filename.asy`. 也可以直接在文档中加上一句: `settings.render=0`

5. `asy -f pdf -render=4 filename.asy` 则是把 OpenGL 渲染出了的图形按 4 份去取图形, 然后作为 pdf 的封面, 当我们用 Adobe pdf 阅读器 9.0 以上版本打开, 然后单击, 才会激活那个真 3D 渲染. `render=4` 可以换成其他数, 得到的图形质量会提高一些.

3.3 关于 3D 投影

画 3D 图形与画 2D 的区别在于, 对 3D 图形, 我们需要一种把 3D 图形的信息投影到 2D 的方式, 因为毕竟我们的输出设备都是 2D 的东西.

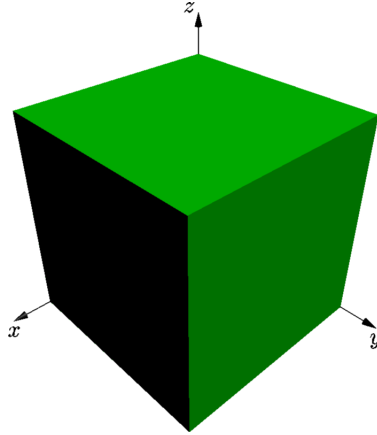
3.3.1 透视投影

所谓透视投影, 就是假想有一个称为相机 (camera) 的东西, 在相对于目标 (target) 的某一个位置观察该物体. 这种观察物体的方式称为 `perspective` 函数. 有下面两种等价的引用办法.

```
perspective(triple camera, triple up=Z, triple target=0)
```

```
perspective(real x, real y, real z, triple up=Z, triple target=0)
```

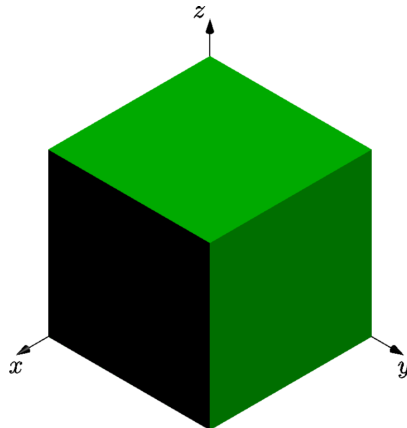
默认投影 `currentprojection` 的初始值为 `perspective(5,4,2)`.



```
import graph3;
currentprojection=perspective(1,1,1);
size(200);
limits((0,0,0),(1.2,1.2,1.2));
draw(unitcube,green);
axes3("$x$","$y$","$z$",Arrow3);
```

3.3.2 正交投影

从三维到二维的正交投影, 又称为正投影, 直角投影等. 该投影假想一束沿着某个三维向量方向从无穷远的处照射过来的光线 (比如太阳的平行光) 照射过来所得到的影. 这个投影把平行的直线投影到平行直线. 正是因为光线是从无穷远处过来的, 因此正交投影不会因近处的物体看上去大一些, 而远处的物体看上去小一些. 这就是建筑和工程设计通常都是采用保持相对大小和角度的投影.



```
import graph3;
currentprojection=orthographic(1,1,1);
size(200);
limits((0,0,0),(1.2,1.2,1.2));
draw(unitcube,green);
axes3("$x$","$y$","$z$",Arrow3);
```

3.3.3 斜投影

斜投影与正交投影其实都同属于平行投影. 正交投影能够比较好的保持物体的一个面的精确形状, 但无法很好的表现它的立体特性. 斜投影能通常能够表现物体某个面的精确形状, 同时能够表现出物体的三维形状.

oblique

oblique(real angle)

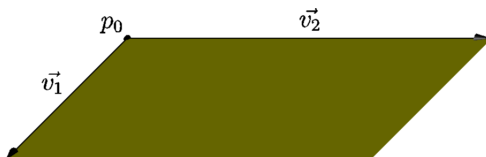
点 (x, y, z) 投影到 $(x - 0.5z, y - 0.5z)$. 如果可选的角度给出, 那么 z 轴的负半轴向下转相应的角度. obliqueZ 与 bolique 是同一个意思.

obliqueX

obliqueY(real angle)

点 (x, y, z) 投影到 $(x + 0.5y, z + 0.5y)$. 如果可选的角度给出, 那么 y 轴的正半轴向下转相应的角度.

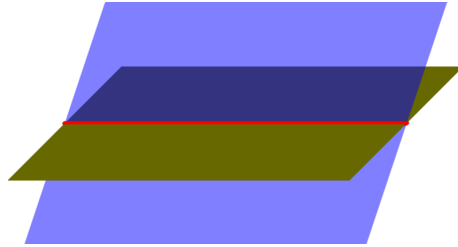
我们可以用斜投影画我们在立体几何里面常见的平面.



```
size(8cm,0);
import three;
currentprojection=obliqueX;
triple v1=(4,0,0),
v2=(0,6,0),
p0=(-2,-3,0);
path3 pl=plane(v1,v2,p0);
draw(surface(pl),yellow);
draw(Label("$\vec{v_1}$"),p0--p0+v1,Arrow3);
draw(Label("$\vec{v_2}$"),p0--p0+v2,N,Arrow3);
dot("$p_0$",p0,NW);
```

其中的 `plane` 是一个利用 `p0` 点和向量 `v1`, `v2` 的画平面 `p0--p0+v1--p0+v1+v2--p0+v2---``cycle` 的函数.

下例是两个平面相交做到自动消隐.



```
size(7.5cm,0);
import three;
currentprojection=obliqueX;
currentlight=light((8,10,2),(8,0,8));
triple v1=(4,0,0),
v2=(0,6,0),
p0=(-2,-3,0);
path3 pl1=plane(v1,v2,p0);
path3 pl2=rotate(45,Y)*pl1;
draw(surface(pl1),yellow);
draw(surface(pl2),blue+opacity(.5));
draw(-v2/2--v2/2,2bp+red);
```

3.4 3D 的数据类型

对应于 2D 情形, 我们在 3D 的情况下有下列数据类型, 我们先简单的介绍一下.

`triple`

有序的三元实数组 (x, y, z) , 如果一个三元组为 `v`, 则它的各个分量分别用 `v.x`, `v.y` 和 `v.z` 引用. 它的默认初始值为 $(0.0, 0.0, 0.0)$.

`path3`

三维路径, 与二维一样, 有 `..` 和 `--` 两种连接点的方式.

`guide3`

待画出的三维路径.

`surface`

曲面, 与二维的 `graph` 对应.

`transform3`

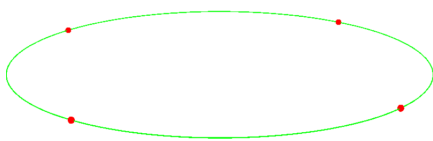
三维变换, 包括 `shift`(平移), `xscale3`, `yscale3`, `zscale3`, `scale3`(各种放大), `rotate`(旋转), `reflect`(反射).

`revolution`

旋转体, 而且包含了旋转体经线, 纬线, 轮廓等信息.

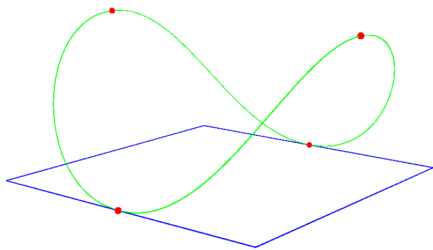
3.5 3D 曲线

我们先看一个在 XY 平面上画曲线的情形, 此时 z 坐标都是 0. 下面是圆在三维空间中看起来的模样.



```
import three;
size(200);
path3 g=(1,0,0)..(0,1,0)..(-1,0,0)..(0,-1,0)..cycle;
draw(g,green);
dot(g,red);
```

我们把其中第二和第四两个点的 z 坐标从 0 变到 1, 这时就得到是空间中扭曲的曲线.

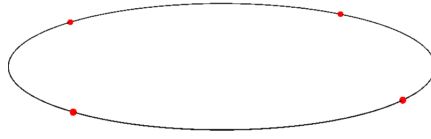


```
import three;
size(200);
path3 g=(1,0,0)..(0,1,1)..(-1,0,0)..(0,-1,1)..cycle;
draw(g,green);
dot(g,red);
draw((-1,-1,0)--(1,-1,0)--(1,1,0)--(-1,1,0)--cycle,blue);
```

Asymptote 定义了一个与数据类型 `path3` 名字一样 `path3` 的函数用来把平面曲线转成三维曲线.

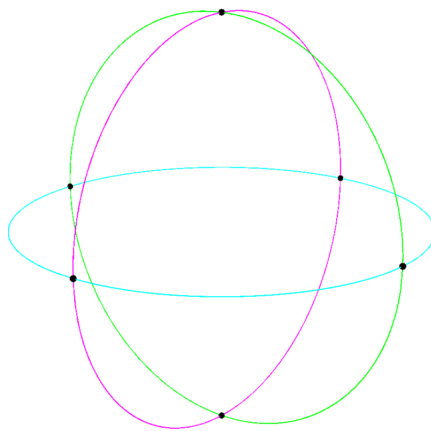
```
path3 path3(path p, triple plane(pair)=XYplane);
```

刚才的 XY 平面的圆也可以如下画出.

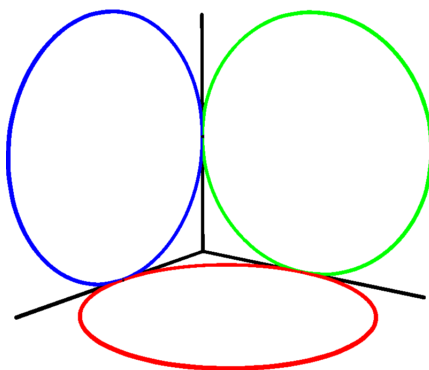


```
import three;
size(200);
path p=(1,0)..(0,1)..(-1,0)..(0,-1)..cycle;
path3 p3=path3(p);
draw(p3);
dot(p3,red);
```

默认是把平面曲线提升为 XY 平面上的曲线, 也可以换成是 YZ 和 ZX 平面中的曲线.

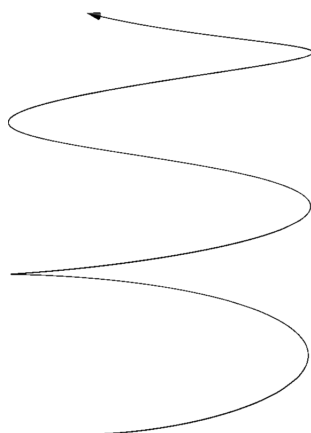


```
import three;
size(200);
path p=(1,0)..(0,1)..(-1,0)..(0,-1)..cycle;
path3 pXY=path3(p,XYplane);
path3 pYZ=path3(p,YZplane);
path3 pZX=path3(p,ZXplane);
draw(pXY,cyan);
draw(pYZ,green);
draw(pZX,magenta);
dot(pXY);
dot(pYZ);
dot(pZX);
```

```
import graph3;
size(200);
defaultpen(linewidth(2));
path p=(1,0)..(0,1)..(-1,0)..(0,-1)..cycle;
draw(shift(X+Y)*XY()*path3(p),red);
draw(shift(Y+Z)*YZ()*path3(p),green);
draw(shift(Z+X)*ZX()*path3(p),blue);
axes3();
```

如果给出参数曲线的方程, 我们可以通过 `graph` 函数把该参数方程转成 `path3`, 因此要调用 `graph3` 宏包.

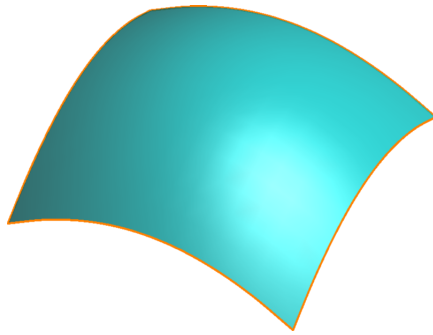


```
import graph3;
size(0,200);
real x(real t) {return cos(2pi*t);}
real y(real t) {return sin(2pi*t);}
real z(real t) {return t;}
path3 p=graph(x,y,z,0,2.7,operator ..);
draw(p,Arrow3);
```

这是一条螺旋上升的空间线.

3.6 Bezier 曲面片

Asymptote 的曲面都是由若干张称为 Bezier 补丁的曲面片拼接而成的. 这个 Bezier 补丁的原理与 Bezier 曲线的原理是类似的, 都是利用控制点来生成曲面的, 因此与曲线一样, 都是逐点的, 这非常有利于实现逐点着色的效果. 我们详细地讨论一下这种 Bezier 补丁, 因为我们的许多 3D 的效果都是基于我们对它的认识.



```
import three;
currentprojection=orthographic(2,-4,5);
size(200);
triple[] [] P={
    {(-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5)},
    {(-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0)},
    {(0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0)},
    {(1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5)}
};
surface sf=surface(patch(P));
draw(surface(sf),cyan+opacity(0.8));
draw(sequence(new path3(int i){
return sf.s[i].external();},sf.s.length), orange+1bp);
```

若干张 Bezier 曲面就可以拼接成我们想要的图形. 我们看到如下四个 Bezier 曲面片拼成一个茶壶把手, 至于它们能无缝地在公共边界相接, 主要是得益于控制点. 下面也演示了如何独立的在各个曲面片独立的着色. Asymptote 还支持逐点着色, 这个将在后面介绍.



```

import three;
size(200);
triple[] [] P={
    {(-2.7,0,1.65),(-2.7,0.3,1.65),(-3,0.3,1.65),(-3,0,1.65)},
    {(-2.7,0,1.875),(-2.7,0.3,1.875),(-3,0.3,2.1),(-3,0,2.1)},
    {(-2.3,0,1.875),(-2.3,0.3,1.875),(-2.5,0.3,2.1),(-2.5,0,2.1)},
    {(-1.6,0,1.875),(-1.6,0.3,1.875),(-1.5,0.3,2.1),(-1.5,0,2.1)}},

    {(-1.6,0,1.875),(-1.6,-0.3,1.875),(-1.5,-0.3,2.1),(-1.5,0,2.1)},
    {(-2.3,0,1.875),(-2.3,-0.3,1.875),(-2.5,-0.3,2.1),(-2.5,0,2.1)},
    {(-2.7,0,1.875),(-2.7,-0.3,1.875),(-3,-0.3,2.1),(-3,0,2.1)},
    {(-2.7,0,1.65),(-2.7,-0.3,1.65),(-3,-0.3,1.65),(-3,0,1.65)}},

    {(-2.7,0,1.65),(-2.7,-0.3,1.65),(-3,-0.3,1.65),(-3,0,1.65)},
    {(-2.7,0,1.425),(-2.7,-0.3,1.425),(-3,-0.3,1.2),(-3,0,1.2)},
    {(-2.5,0,0.975),(-2.5,-0.3,0.975),(-2.65,-0.3,0.7275),(-2.65,0,0.7275)},
    {(-2,0,0.75),(-2,-0.3,0.75),(-1.9,-0.3,0.45),(-1.9,0,0.45)}},

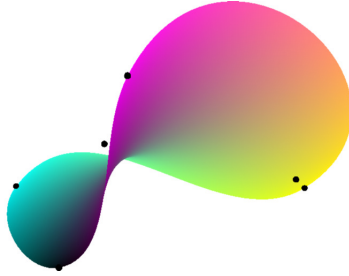
    {(-2,0,0.75),(-2,0.3,0.75),(-1.9,0.3,0.45),(-1.9,0,0.45)},
    {(-2.5,0,0.975),(-2.5,0.3,0.975),(-2.65,0.3,0.7275),(-2.65,0,0.7275)},
    {(-2.7,0,1.425),(-2.7,0.3,1.425),(-3,0.3,1.2),(-3,0,1.2)},
    {(-2.7,0,1.65),(-2.7,0.3,1.65),(-3,0.3,1.65),(-3,0,1.65)}}
};
pen[] colors={cyan,yellow,magenta,green};
for(int i=0; i < P.length; ++i) {
    draw(surface(patch(P[i])),colors[i]);
}

```

对 Bezier 曲面, Asymptote 提供了一些函数, 我们观察其中一个 `surface` 函数

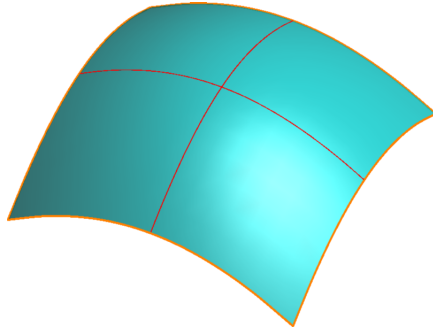
```
surface surface(path3 external, triple[] internal=new
triple[],triple[] normals=new triple[],pen[] colors=new pen[]);
```

该函数提供了给出一条封闭路径搞成一个 Bezier 曲面的办法.



```
import three;
size(200);
path3 p=(1,2,2)..(3,9,2)..(4,4,4)..(2,3,0)..cycle;
surface sf=surface(p,new pen[]{cyan,yellow,magenta,black});
draw(sf,nolight);
dot(sf.s[0].internal());
dot(sf.s[0].external());
```

下面再来看我们如何从 Bezier 补丁返回我们需要的一些曲面上的元素.



```
import three;
currentprojection=orthographic(2,-4,5);
size(200);
triple[] [] P={
  {(-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5)},
  {(-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0)},
  {(0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0)},
  {(1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5)}
};
```

```

surface sf=surface(patch(P));
draw(sf,cyan+opacity(0.8));
draw(sf.s[0].uequals(0.5),red);
draw(sf.s[0].vequals(0.5),red);
draw(sequence(new path3(int i){
    return sf.s[i].external();},sf.s.length), orange+1bp);

```

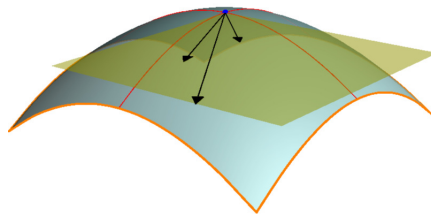
其中 `triple[] [] P` 给出了生成该 Bezier 补丁的 4 个点, `surface` 函数使得它变成数据类型为 `surface`.

`sf.s[0]` 返回的是这个曲面的第 0 块 Bezier 补丁.(因为我们现在只有一片补丁, 该补丁标号为 0)

`sf.s[0].uequals` 和 `sf.s[0].vequals` 就是 Bezier 补丁上的坐标线 (u-曲线和 v-曲线). 此处画了区中的两条 (0.5). 由此可以看出, 我们可以访问该补丁上任何一个点.

`sf.s[i].external()` 返回的是第 *i* 个 Bezier 补丁边界的路径. `sf.s.length` 是 Bezier 补丁的个数.

我们利用上面介绍的性质画出 Bezier 曲面某个点处的切平面.



```

import three;
size(200,0);
triple[] [] P={
    {(-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5)},
    {(-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0)},
    {(0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0)},
    {(1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5)}
};
surface sf=surface(patch(P));
draw(sf,palecyan+opacity(0.8));
draw(sequence(new path3(int i){
    return sf.s[i].external();},sf.s.length), orange+1bp);
real t=0.5;
path3 u=sf.s[0].uequals(t);
path3 v=sf.s[0].vequals(t);

```

```

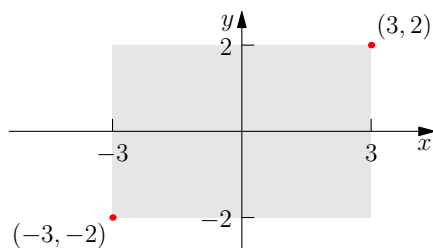
draw(u,red);
draw(v,red);
triple P=point(u,t);
dot(P,blue);
triple dU=dir(u,t);
triple dV=dir(v,t);
draw(surface(plane(1.5dU,1.5dV,P)),yellow+opacity(0.5));
triple dX=0.5dU+dV;
triple dY=0.5dU+0.5dV;
draw(P--P+dX,Arrow3(DefaultHead2(normal=Z)));
draw(P--P+dY,Arrow3(DefaultHead2(normal=Z)));
draw(P--P+dX+dY,Arrow3(DefaultHead2(normal=Z)));

```

3.7 画参数曲面

调用 graph3 宏包, 我们就可以画参数曲面. 此时分几种情况.

如果该参数曲面是以 $z = f(x, y)$ 的所谓显示形式给出的, 例如是 $z = f(x, y) = x^2 - y^2$, 且定义在矩形域上, x 的变化范围是 -3 到 3, y 的变化范围是 -2 到 2, 此时该矩形的左下角坐标是 (-3,-2), 右上角坐标是 (3,2), 即 XY 平面中的 `box(a,b)` 其中 $a=(-3,2)$, $b=(3,2)$ 请见下图.



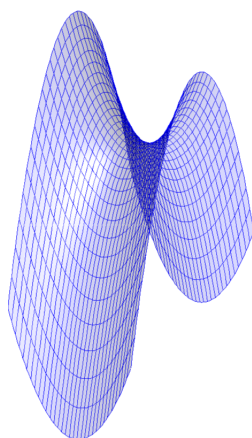
```

size(200);
import graph;
pair A=(-3,-2),B=(3,-2),C=(3,2),D=(-3,2);
fill(A--B--C--D--cycle,lightgray);
label("$(-3,-2)$",A,SW);
label("$ (3,2)$",C,NE);
xtick("$-3$",-3);
xtick("$3$",3);
ytick("$-2$",-2);
ytick("$2$",2);
dot(A,red);
dot(C,red);

```

```
xaxis("$x$",Arrow,above=true);
yaxis("$y$",Arrow,above=true);
```

然后我们就可以画用显式表示的曲面参数方程. 下图



```
import graph3;
size(0,200);
currentprojection=orthographic(3,2,4);
currentlight=(5,3,0);
real f(pair z) {return z.y^2-z.x^2;}
draw(surface(f,(-3,-2),(3,2),nx=32,Spline),lightblue+opacity(0.3),meshpen=blue);
```

上面代码中我们用 $f(t)$ 表示该二元函数, 其中 t 是二元组, 因此二元函数的两个元就分别用 $t.x$ 和 $t.y$ 分别表示, 原来的 $x^2 - y^2$ 现在就表示成 $t.x^2 - t.y^2$ 了.

从 $(-3, -2)$ 和 $(3, 2)$ 这两个二元组, 我们知道现在 $t.x$ 的变动范围是 -3 到 3 , $t.y$ 的变动范围是 -2 到 2 .

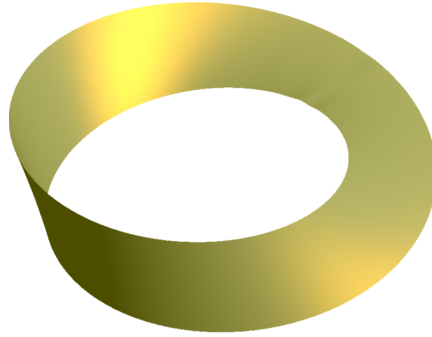
Spline 在两个 Beizer 的拼接处采用光滑的插值过渡, 因此所得到的图形色彩处理会连续. 请把 "Spline" 去掉对比不同的效果.

$nx=32$ 是指把曲面定义域分成 32×32 个小矩形, 从而画出的曲面是由 32×32 片曲面片拼成的.(默认 $ny=nx$)

$meshpen=blue$ 是定义画出网格的画笔颜色.

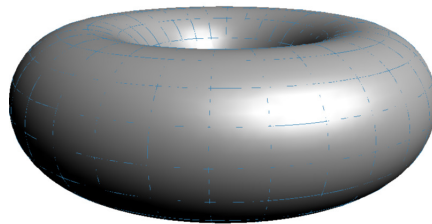
对于依赖于两个参数的一般参数曲面, 则它的方程类似以下形式. 注意, 这是一条著名的 Mobius 带.

$$f(s, t) = \begin{cases} x(s, t) = (R + s \cdot \cos(\frac{t}{2})) \cdot \cos(t); \\ y(s, t) = (R + s \cdot \cos(\frac{t}{2})) \cdot \sin(t); \\ z(s, t) = s \cdot \sin(\frac{t}{2}); \end{cases} \quad (3.1)$$



```
import graph3;
size(200);
currentprojection=orthographic(-4,-4,5);
real R=12;
triple f(pair t){
    real u=t.x;
    real v=t.y;
    real x=(R+u*cos(v/2))*cos(v);
    real y=(R+u*cos(v/2))*sin(v);
    real z=u*sin(v/2);
    return (x,y,z);
}
material m=(material(diffusepen=grey,ambientpen=yellow,
                    emissivepen=black,specularpen=orange));
surface s=surface(f,(-pi,0),(pi,2pi),50,Spline);
draw(s,m);
```

我们也可以采用一个稍微紧凑一点的形式列出参数方程.



```
import graph3;
size(200,0);
real R=2;
real a=1;
triple f(pair t) {
    return ((R+a*cos(t.y))*cos(t.x),(R+a*cos(t.y))*sin(t.x),a*sin(t.y));
}
```



```

}
pen p=rgb(0.2,0.5,0.7);
draw(surface(f,(0,0),(2pi,2pi),30,15,Spline),lightgray,meshpen=p);
//draw(surface(f,(0,0),(2pi,2pi),40,20),nullpen,meshpen=p+thick());

```

其中 `meshpen=p+thick()` 的 `thick()` 是希望画出的网格比较清晰. 而

```

draw(surface(f,(0,0),(2pi,2pi),40,20),nullpen,meshpen=p+thick());

```

则是只是画出那些网格, 并不蒙上表面.

3.8 3D 字母, 坐标轴, 刻度和栅格 (grid)

调用 `graph3`, `grid3` 宏包, 可以比较方便的画出一一般数学图形的坐标轴, 坐标轴上的刻度, 还可以画出一个立体框, 一个三维栅栏以增强立体效果.

在 3D 中画 x 轴采用如下的 `xaxis3` 函数, `yaxis3` 和 `zaxis3` 是类似的.

```

void xaxis3(picture pic=currentpicture, Label L="",axis axis=YZZero,
real xmin=-infinity, real xmax=infinity, pen p=currentpen, ticks3
ticks=NoTicks3, arrowbar3 arrow=None, bool above=false);

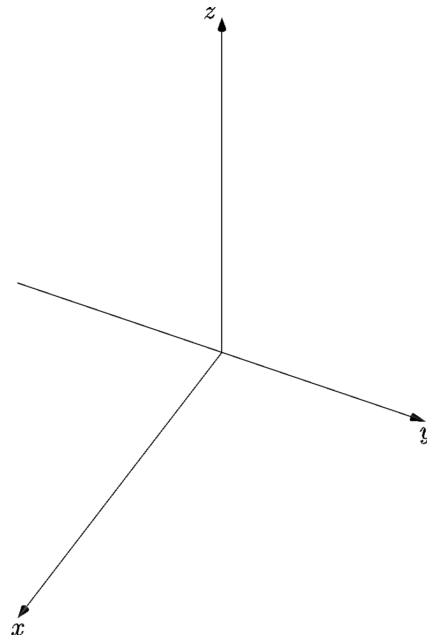
```

也有同时画出所有坐标轴的方式.

```

void axes3(picture pic=currentpicture, Label xlabel="",Label
ylabel="", Label zlabel="", triple
min=(-infinity,-infinity,-infinity), triple
max=(infinity,infinity,infinity), pen p=currentpen, arrowbar3
arrow=None);

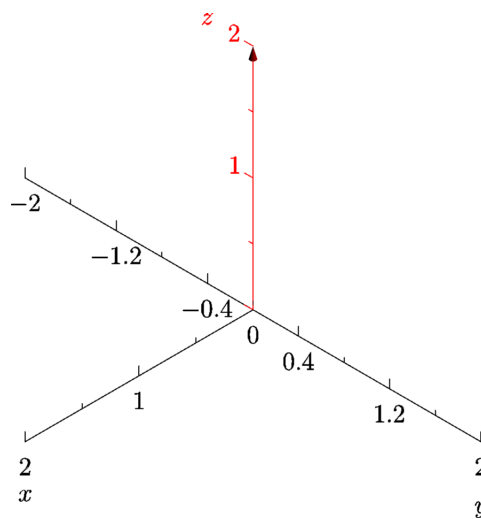
```



```
import graph3;
size(200,0,IgnoreAspect);
currentprojection=orthographic(1,1,1);
limits((0,-2,0), (2,2,2));
axes3("$x$","$y$","$z$",Arrow3);
```

其中 `limits` 给出一个立方体范围, x 从 0 到 2, y 从 -2 到 2, z 从 0 到 2.

3D 刻度的可选项包括 `NoTick3`, `InTicks`, `OutTicks` 以及 `InOutTicks`.



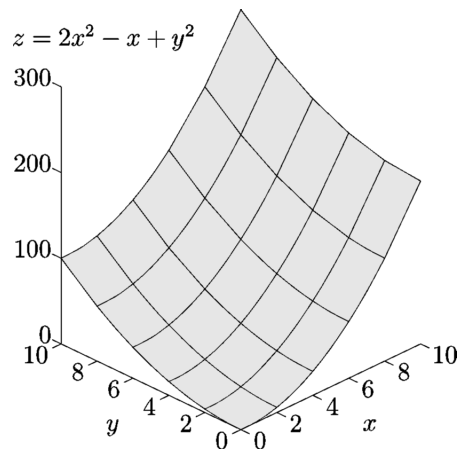
```
import graph3;
size(8cm,0);
```

```

currentprojection=orthographic(1,1,1);
limits((0,-2,0),(2,2,2));
xaxis3("$x$", InTicks());
yaxis3("$y$", InTicks(Label,5,2));
zaxis3("$z$", OutTicks(beginlabel=false),p=red,arrow=Arrow3);

```

InTicks 中的 5 表示标出 5 个区间的刻度, 2 表示每隔两个标一个坐标. beginlabel=false 不把那个原点出的标签标出, 因为与 x 轴的那个混在一起不太美观.



```

import graph3;
import contour;
size(7.5cm,0);
size3(7.5cm,IgnoreAspect);
real f(pair z) {
    return 2z.x^2-z.x+z.y^2;
}
currentprojection=orthographic(-10,-10,200);
limits((0,0,0),(10,10,300));
xaxis3(Label("$x$",position=MidPoint,align=SE),OutTicks(Step=2));
yaxis3(Label("$y$",position=MidPoint,align=SW),OutTicks(Step=2));
zaxis3(Label("$z=2x^2-x+y^2$",position=EndPoint,align=3N+E),
        Bounds(Min,Max),InTicks(Step=100,Label(align=Y)));
draw(surface(f,(0,0),(10,10),nx=5,Spline),lightgray,meshpen=black+thick(),nolight);

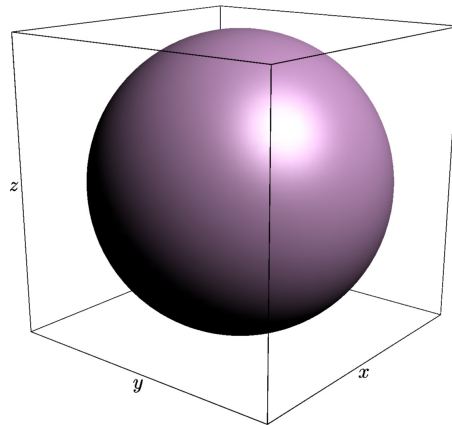
```

graph3 宏包还提供了一个 Bounds 函数来画出一个边框来显示图形的定界.

```

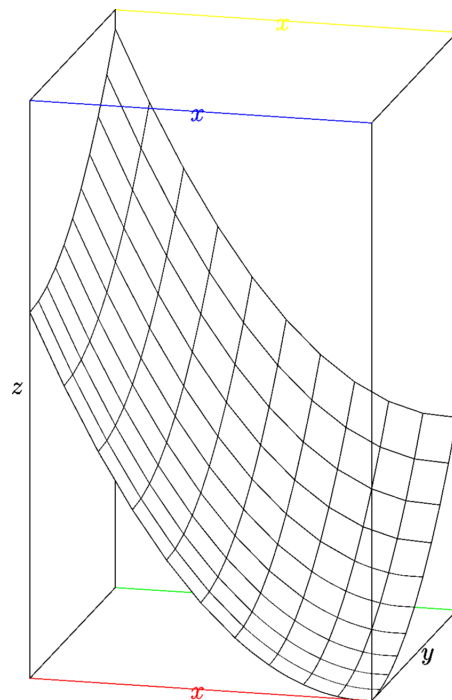
axis Bounds(int type=Both, int type2=Both, triple align=0, bool extend=false);

```



```
import graph3;
size3(200);
draw(unitsphere,pink);
xaxis3("$x$",Bounds());
yaxis3("$y$",Bounds());
zaxis3("$z$",Bounds());
```

`Bounds` 接受两种类型的参数, 这些参数是在 `Min`, `Max`, `Both` 中选取. 这两个参数用来控制与某个坐标轴平行的四条盒子边界是否画出.



```
import graph3;
size(7.5cm,0);
```

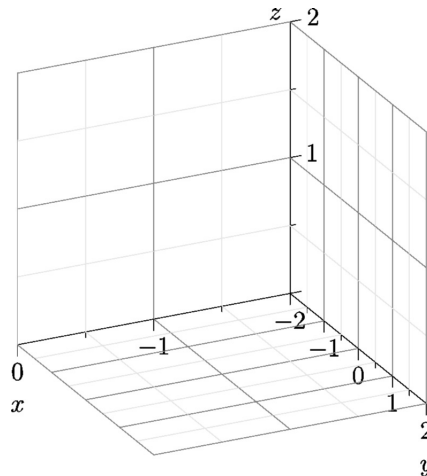
```

size3(7.5cm,7.5cm,12cm,IgnoreAspect);
real f(pair z) {
    return 2z.x^2+z.x+z.y^2;
}
currentprojection=orthographic(5,-20,100);
limits((-10,0,0),(0,10,300));
xaxis3(Label("$x$"),Bounds(Min,Min),red);
xaxis3(Label("$x$"),Bounds(Max,Min),green);
xaxis3(Label("$x$"),Bounds(Min,Max),blue);
xaxis3(Label("$x$"),Bounds(Max,Max),yellow);
yaxis3(Label("$y$"),Bounds());
zaxis3(Label("$z$"),Bounds());
draw(surface(f,(-10,0),(0,10),nx=10,Spline),
    lightgray+white,meshpen=black+thick(),nolight);

```

从上面例子可以看出, `Bounds(Min,Min)` 画出过 `align=0` 点的那个坐标轴, `Bounds(Max,Min)` 与 `Bounds(Min,Max)` 是分别沿着 `Y` 和 `Z` 方向移动到边界, `Bounds(Max,Max)` 则是同时往 `Y` 和 `Z` 方向移动到最外面. 这里是针对 `xaxis3` 论述 `Max` 和 `Min` 的涵义, 其他的 `yaxis3` 和 `zaxis3` 是类似的.

Asymptote 的一个 `grid3` 宏包提供了画三维栅格的功能, 效果如下:



```

import grid3;
size(200,0,IgnoreAspect);
currentprojection=orthographic(0.25, 1, 0.25);
limits((-2,-2,0),(0,2,2));
grid3(pic=currentpicture,gridroutine=XYZgrid(pos=Relative(0)),
    N=0,n=0,Step=0,step=0,begin=true,end=true,pGrid=grey,pgrid=lightgrey,
    above=false);
//grid3(XYZgrid);

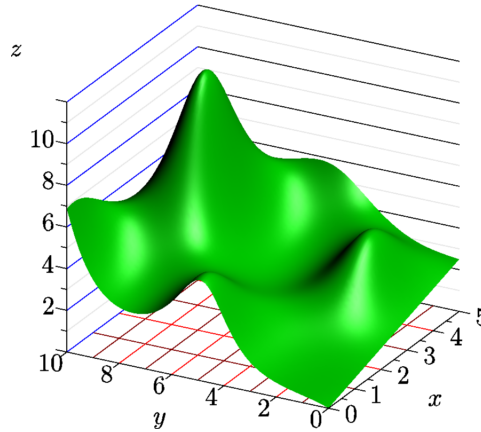
```

```

xaxis3(Label("$x$"),position=EndPoint,align=S), Bounds(Min,Min), OutTicks());
yaxis3(Label("$y$"),position=EndPoint,align=S), Bounds(Min,Min), OutTicks());
zaxis3(Label("$z$"),position=EndPoint,align=(0,0.5)+W),
      Bounds(Min,Min), OutTicks(beginlabel=false));

```

上面 `grid3` 里面大部分都是默认选项, 因此一般用 `grid3(XYZgrid)` 已经可以生成相同的图形. 下面给出一些综合的例子.



```

import graph3;
import contour;
import grid3;
import palette;
size(8cm,7cm,IgnoreAspect);
currentprojection=orthographic(-10,-10,12);
limits((0,0,0),(5,10,12));
real f(pair z) {return (z.x+z.y)/(2+cos(z.x)*sin(z.y));}
surface s=surface(f,(0,0),(5,10),50,Spline);
draw(s,green);
grid3(new grid3routines [] {XYZgrid, ZXgrid(10), ZYgrid(5)},
      Step=2,
      step=1,
      pGrid=new pen[] {red, blue, black},
      pgrid=new pen[] {0.5red, lightgray, lightgray});
xaxis3(Label("$x$"),position=MidPoint,align=SE),
      Bounds(Min,Min),
      OutTicks());
yaxis3(Label("$y$"),position=MidPoint,align=SW),
      Bounds(Min,Min),
      OutTicks(Step=2));

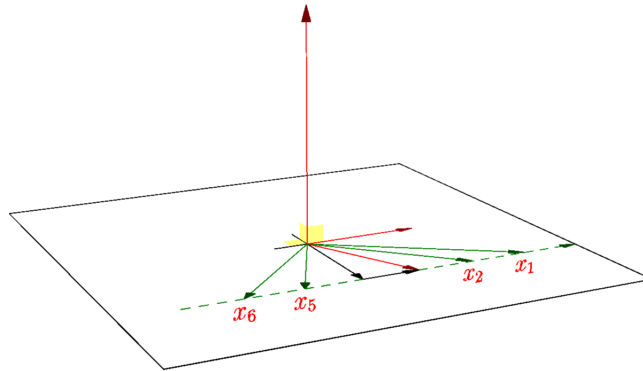
```

```

zaxis3(Label("$z$"),position=EndPoint,align=N+W),
      XEequals(0,10),
      InTicks(beginlabel=false,endlabel=false,Label(align=Y))
    );

```

下例是利用数组的一些性质来标一系列 3D 字母.



```

size(300);
import three;
currentprojection=perspective(20,-10,8);
triple a_orth=2X;
triple b=2Y;
triple a_par=0.5*b;
triple a=a_orth+a_par;
draw((4,4,0)--(-4,4,0)--(-4,-4,0)--(4,-4,0)--cycle);
draw(surface(0--0.3Y--(0.3Y+0.3Z)--0.3*Z--cycle),lightyellow,nolight);
draw(surface(0--(-0.3)*X--(-0.3X+0.3Z)--0.3Z--cycle),lightyellow,nolight);
draw(surface(0--(-0.3Y)--(-0.3Y-0.3X)--(-0.3X)--cycle),lightyellow,nolight);
draw(0--a_orth,Arrow3(DefaultHead2(normal=Z)));
draw(0--b,red,Arrow3(DefaultHead2(normal=Z)));
draw(0--a,red,Arrow3(DefaultHead2(normal=Z)));
draw(0--cross(a,b),red,Arrow3(DefaultHead2()));
draw(a_orth--a_orth+a_par,Arrow3(DefaultHead2(normal=Z)));
draw(0--(-0.3)*b);
draw(0--(-0.3)*a_orth);
draw(a--a+3*a_par,deepgreen+dashed,Arrow3(DefaultHead2(normal=Z)));
draw(a_orth--a_orth+(-3)*a_par,deepgreen+dashed);
int[] I={-2,-1,0,1,2,3};
I=reverse(I);
for(int i: I){

```

```

if(i!=0 && i!=1)
    draw(Label(YZ()*format("$x_{%d}$",find(I==i)+1),EndPoint,align=S,red),
        0--a_orth+i*a_par, deepgreen, Arrow3(DefaultHead2(normal=Z)));
}

```

3.9 3D 变换

3D 对象可以进行一些变换, 它们包括如下数据类型称为 `transform3` 的一些变换.

`shift(triple v)`

依照向量 v 平移;

`xscale3(real x)`

依照 x 轴方向放大 x ;

`yscale3(real y)`

依照 y 轴方向放大 y ;

`zscale3(real z)`

依照 z 轴方向放大 z ;

`scale3(real s)`

依照 x, y , 以及 z 方向放大 s ;

`scale3(real x, real y, real z)`

依照 x 轴, y 轴, z 轴分别放大 x, y, z

`rotate(real angle, triple v)`

绕着以原点为起点的方向 v 旋转 $angle$ 角度.

`rotate(real angle, triple u, triple v)`

绕着轴 $u-v$ 旋转 $angle$ 角度.

`reflect(triple u, triple v, triple w)`

关于 u, v , 和 w 所确定的平面反射.

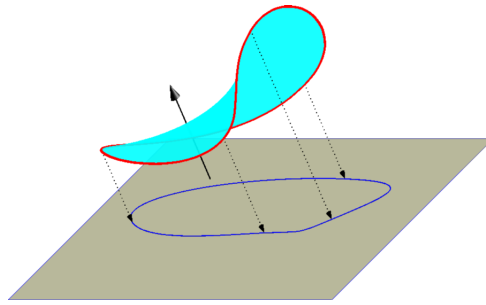
一个重要的 `transform3` 是 `planeproject`.

```

transform3 planeproject(triple n, triple O=0, triple dir=n);
transform3 planeproject(path3 p, triple dir=normal(p));

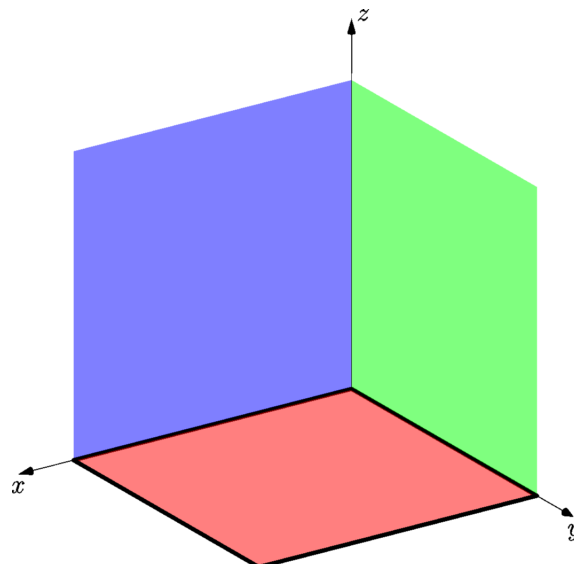
```

第一个变换是沿着 `dir` 方向往那个过 O 并且以 n 为法向量的平面作投影. 第二个变换是给定一个封闭的路径 p 确定的平面, 然后沿着 `dir` 方向投影. 其中 `normal(p)` 返回该封闭路径的法向量, 用来作为 `dir` 的默认方向. 如果 p 不是平面, 就返回 $(0,0,0)$.



```
import three;
size(8cm,0);
currentprojection=obliqueX;
currentlight=(0,2,1);
path3 plane=plane(12X,12Y,(-3,-3,0));
triple dir=(1,-1,4);
path3 p=(5,3,4)..(5,4,8)..(1,4,4)..(4,-2,3)..cycle;
draw(surface(plane),lightyellow+opacity(.5),blue);
draw(surface(p),cyan+opacity(.9),red+1pt,nolight);
transform3 proj=planeproject(plane,dir);
draw(proj*p,dir,blue);
draw(0--dir,Arrow3);
for(int i=0; i<length(p); ++i){
    draw(point(p,i)--point(proj*p,i),dotted,Arrow3(4));
}
```

在 `three.asy` 里面还预先定义了一些数据类型同样是 `transform3` 的投影变换 `XY,YZ,ZX,YX,ZY,ZX`. 当然, 它们同样也可以是函数 `XY()` 等等.

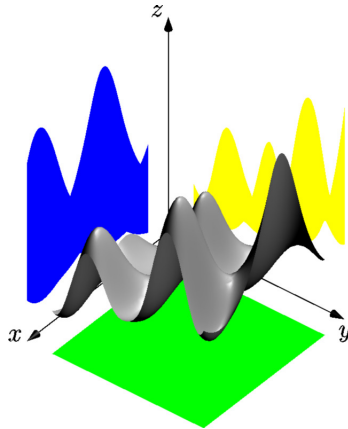


```

import graph3;
size3(200);
currentprojection=orthographic(4,6,3);
currentlight=nolight;
limits((0,0,0),(1.2,1.2,1.2));
draw(unitsquare3,linewidth(2pt));
draw(surface(XY*unitsquare3),red+opacity(0.5));
draw(surface(YZ*unitsquare3),green+opacity(0.5));
draw(surface(ZX*unitsquare3),blue+opacity(0.5));
axes3("$x$","$y$","$z$",Arrow3);

```

利用 `planeproject` 函数, 我们可以画通常的三视图.

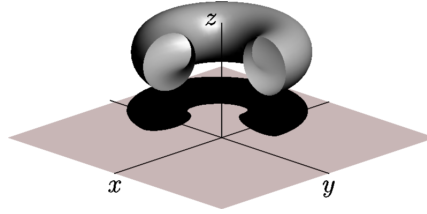


```

import graph3;
size(200);
currentprojection=orthographic(10,8,10);
real f(pair z) {
    real u=z.x, v=z.y;
    return (u/2+v)/(2+cos(u/2)*sin(v));
}
surface s=surface(f,(2,2),(14,14),Spline);
draw(s,palegrey);
draw(planeproject(XY*unitsquare3)*s,green,nolight);
draw(planeproject(YZ*unitsquare3)*s,yellow,nolight);
draw(planeproject(ZX*unitsquare3)*s,blue,nolight);
axes3("$x$","$y$","$z$",Arrow3());

```

也可以利用这个 `planeproject` 函数画出在当前光照光线下立体图形的阴影.



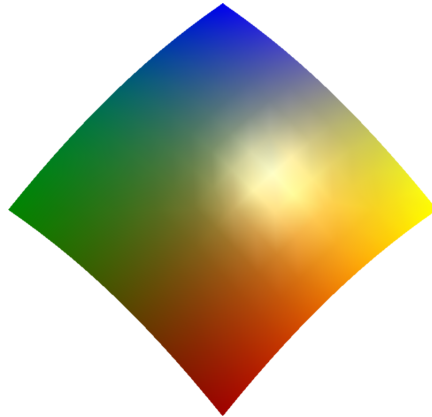
```
import graph3;
currentprojection=orthographic(2,2,1);
size(200);
real R=2;
real a=1;
triple f(pair t) {
    return ((R+a*cos(t.y))*cos(t.x), (R+a*cos(t.y))*sin(t.x), a*sin(t.y));
}
surface s=shift(3Z)*surface(f, (radians(90),0), (radians(345),2pi), 8,8, Spline);
draw(s, lightgrey);
path3 pl=path3((-5,-5)--(5,-5)--(5,5)--(-5,5)--cycle);
draw(shift((-1e-3)*Z)*surface(pl), palered+opacity(0.5));
draw(planeproject(pl, currentlight.position[0])*s);
axes3("$x$", "$y$", "$z$");
```

3.10 着色处理和改变背景颜色

前面已经看到, 给出三维空间中凸的封闭路径, 可以在其中填颜色. 比如

```
draw(surface(path3(polygon(5)), red));
draw(surface(unitcircle(3), red));
draw(surface(unitcircle(3), new pen[] {red, green, blue, black}));
```

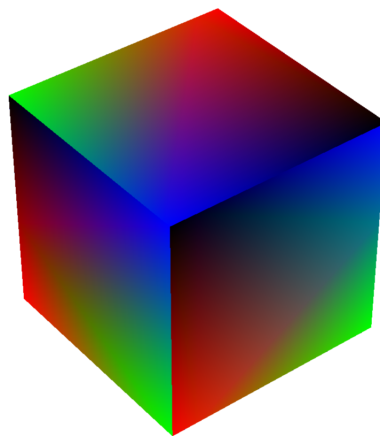
其中, 最后一项匿名函数 new 新建一个颜色数组, 这个数组最多必须恰好包含四种颜色, 代表 Bezier 曲面的四个顶点的初始着色, 然后逐渐向四周变化.



```
import three;
currentprojection=orthographic(1,1,5);
size(200);
triple[] [] P={
    {(-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5)},
    {(-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0)},
    {(0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0)},
    {(1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5)}
};
draw(surface(patch(P,new pen[]{blue,green,red,yellow})));
```

Asymptote 一般的曲面是由许许多多的这种 Bezier 曲面按照一定规则拼接和安置着色, 就做到整体的渐变着色, 这种整体的效果就是我们在前面已经看到过的.

我们可以对每一个这种 Bezier 补丁孤立的着色, 以达到另外一种效果.

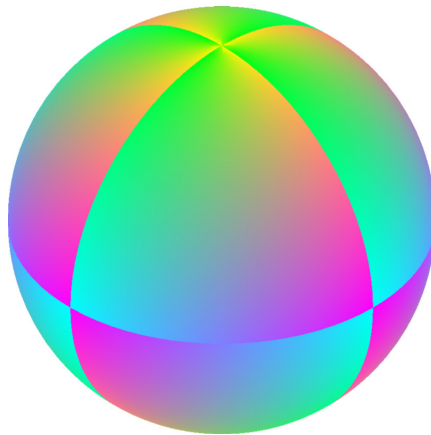


```
import three;
size(200);
currentprojection=perspective(4,5,5);
```

```

real a=1,b=1,c=1;
path3[] g={
  (0,0,0)--(a,0,0)--(a,b,0)--(0,b,0)--cycle,
  (0,0,0)--(a,0,0)--(a,0,c)--(0,0,c)--cycle,
  (a,0,0)--(a,b,0)--(a,b,c)--(a,0,c)--cycle,
  (a,b,0)--(0,b,0)--(0,b,c)--(a,b,c)--cycle,
  (0,b,0)--(0,0,0)--(0,0,c)--(0,b,c)--cycle,
  (0,0,c)--(a,0,c)--(a,b,c)--(0,b,c)--cycle
};
pen[] pen={red,green,blue,black};
for(path3 p:g){
  draw(surface(p,pen),nolight);
}

```

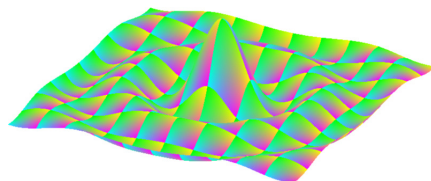


```

import graph3;
size(200);
currentprojection=orthographic(4,4,4);
surface sf=unitsphere;
for(int i=0;i<sf.s.length;++i){
  draw(surface(sf.s[i].external(),sf.s[i].internal(),
    new pen[] {cyan,magenta,yellow,green}),nolight);
}

```

上面我们利用 `sf.s[i]` 返回生成球面 `sf` 的所有 Bezier 补丁, 然后进行着色处理.

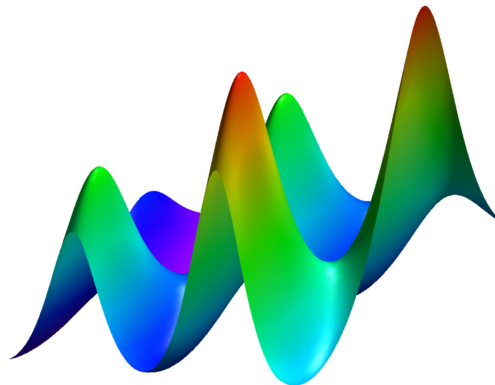


```

import graph3;
currentprojection=orthographic(1,-2,1);
currentlight=(1,-1,0.5);
size(200,0);
real sinc(pair z) {
    real r=2pi*abs(z);
    return r != 0 ? sin(r)/r : 1;
}
surface sf=surface(sinc,(-2,-2),(2,2),Spline);
pen[] p= {cyan,magenta,yellow,green};
for(int i=0;i<sf.s.length;++i){
    draw(surface(sf.s[i].external(),sf.s[i].internal()),p,nolight);
}

```

Asymptote 可以自定义着色函数方案, 然后调用 `palette` 去着色. 我们常用的着色函数包括沿着 z 坐标轴的方向的 `zpart` 函数 (或 x 或 y 轴方向的 `xpart` 和 `ypart`), 以及沿着径向的 `abs` 函数.

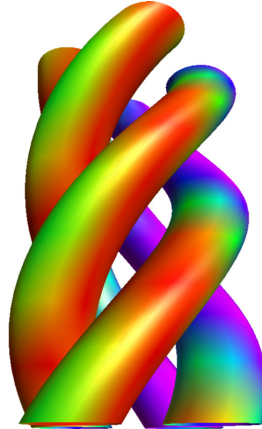


```

import graph3;
import palette;
size(200,IgnoreAspect);
currentprojection=orthographic(1,0.8,1);
real f(pair z) {
    real u=z.x, v=z.y;
    return (u/2+v)/(2+cos(u/2)*sin(v));
}
surface s=surface(f,(2,2),(14,14),40,20,Spline);
s.colors(palette(s.map(zpart),Rainbow()));
draw(s);

```

其中 `s.map()` 接受一个与 `triple` 相关的实数值函数, 上面是 `zpart` 函数.



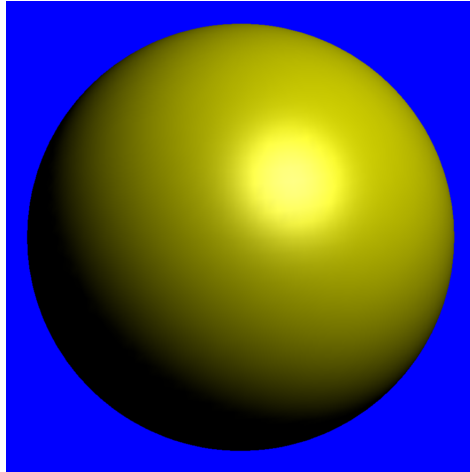
```
import graph3;
import palette;
size(200,200,keepAspect=true);
real w=0.4;
real f(triple t) {return sin(t.x);}
triple f1(pair t) {return (cos(t.x)-2cos(w*t.y),sin(t.x)-2sin(w*t.y),t.y);}
triple f2(pair t) {return (cos(t.x)+2cos(w*t.y),sin(t.x)+2sin(w*t.y),t.y);}
triple f3(pair t) {return (cos(t.x)+2sin(w*t.y),sin(t.x)-2cos(w*t.y),t.y);}
triple f4(pair t) {return (cos(t.x)-2sin(w*t.y),sin(t.x)+2cos(w*t.y),t.y);}
surface s1=surface(f1,(0,0),(2pi,10),8,8,Spline);
surface s2=surface(f2,(0,0),(2pi,10),8,8,Spline);
surface s3=surface(f3,(0,0),(2pi,10),8,8,Spline);
surface s4=surface(f4,(0,0),(2pi,10),8,8,Spline);
s1.colors(palette(s1.map(f),Rainbow()));
s2.colors(palette(s2.map(f),Rainbow()));
s3.colors(palette(s3.map(f),Rainbow()));
s4.colors(palette(s4.map(f),Rainbow()));
draw(s1);
draw(s2);
draw(s3);
draw(s4);
```

这里先定义了一个与 `triple` 相关的实数值函数 `f=sin(t.x)`, 这种函数就像是一个与空间位置相关的温度分布函数那样.

`s.map()` 接受 `f` 这个与 `triple` 相关的实数值函数, 这时是按照 `sin(t.x)` 的函数关系去改变颜色.

另外, 有时候我们希望改变那个基于 OpenGL 的渲染得到的图形的背景颜色, 这时可以用下面的方式.

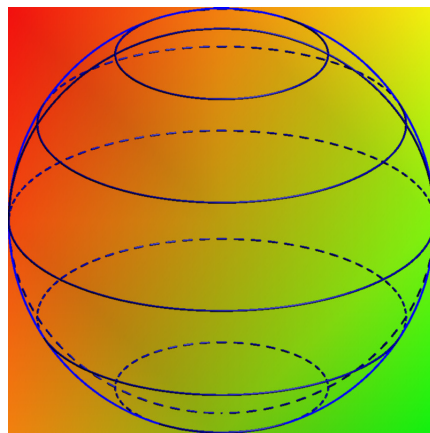
```
currentlight.background=pen;
```



```
size(200);  
import three;  
currentlight.background=blue+opacity(0.1);  
viewportmargin=(10,10);  
currentprojection=orthographic(5,4,3);  
draw(unitsphere,yellow);
```

而 `viewportmargin=(10,10);` 使得视图的水平和竖直方向有一定的空隙, 用以突出背景.

下面提供了另外一种方法, 可以使得背景可以如同曲面那样着色.



```
size(200);  
import solids;
```



```

currentprojection=orthographic(5,4,3);
revolution Sp=sphere(1);
draw(Sp.silhouette(100),blue+1pt);
draw(Sp,m=7,blue+1pt);
currentpicture.add(new void(frame f, transform3 t, picture pic, projection P) {
    draw(f,surface(invert(box(min(f,P),max(f,P)),min3(f),P),
        new pen[] {orange,green,yellow,red}));
});

```

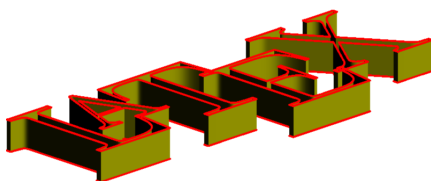
3.11 柱状图形与立体字体

`extrude` 函数可以画出柱状曲面.

```

surface extrude(path p, triple elongation=Z)

```



```

import three;
size(200);
currentprojection=orthographic(-2,-2,1);
path[] g=texpath("\LaTeX");
for(path p:g){
    draw(path3(p),red+1pt);
    draw(extrude(p,2Z),yellow);
    draw(shift(2Z)*path3(p),red+1pt);
}

```

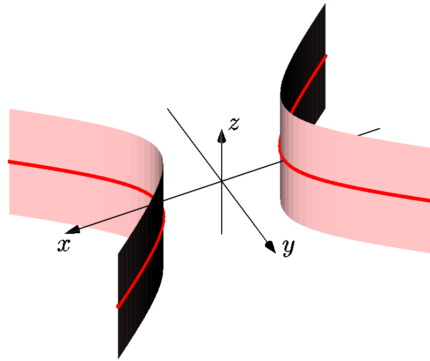
我们用 `texpath` 函数提取了 \LaTeX 徽标的路径, `g` 于是由若干条 `path` 构成. 接下去我们用 `path p` 去遍历 `g` 的所有路径, 用的 Java 风格的循环方式. `path3(p)` 把 2D 路径 `p` 变为 3D 中 XY 平面路径, 这个用法我们前面已经见过, 接下去的 `extrude(p, 2Z)` 把 2D 路径 `p` 沿着 `2Z` 方向突出, 从而画出侧面.

利用这个 `extrude` 函数, 我们可以画出椭圆柱面, 抛物柱面, 双曲柱面等图形, 下面以双曲柱面为例.

双曲柱面的方程是 $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$, 即对表面上的点 (x, y, z) , (x, y) 要满足上述关系, z 可以任意. 相当于把 XY-平面的双曲线上下平移扫出的曲面. 我们可以不用手工把双曲线的方程化成参数方程形式也可以得到它的图形, 这只需调用 `contour` 宏包.

```
guide[] [] contour(real f(real, real), pair a, pair b, real[] c,
    int nx=ngraph, int ny=nx, interpolate join=operator --);
```

画出函数 $f(x,y)$ 的等值线, 等的是数组 `real[] c` 的值, 并且只取 `box(a,b)` 这个矩形限定的部分, `nx` 和 `ny` 分别指定了 `box(a,b)` 的横向与纵向分割的份数. 为了得到更光滑的效果, 我们可以把它们适当调大一些.

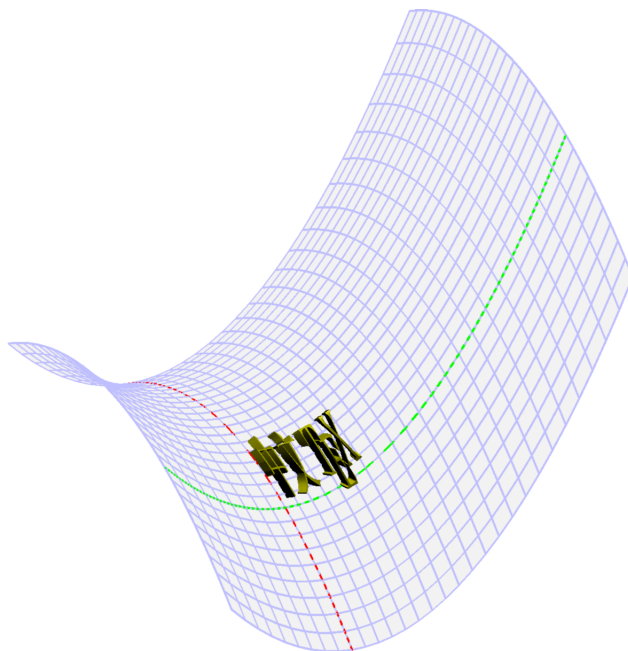


```
import contour;
import graph3;
currentprojection=orthographic(5,10,10);
currentlight=(10,10,2);
size(200);
real a=4;
real b=3;
real f(real x, real y){return x^2/a^2-y^2/b^2;}
guide[] [] g=contour(f,(-10,-10),(10,10),new real[] {1},50);
for(path p:g[0]){
    draw(extrude(p,4Z),palered);
    draw(extrude(p,-4Z),palered);
    draw(path3(p),red+2pt);
}
axes3("$x$","$y$","$z$",Arrow3());
```

因为我们要画 $f(x,y)$ 等 1 那个值的曲线, 因此我们用匿名函数 `new real[] {1}` 临时定义了一个数组 `{1}`. 接下去利用这个曲线和 `extrude` 函数分别向 `4Z`, `-4Z` 两个方向突出.

1.84 以后的版本利用 `extrude` 函数实现了在曲面上标各种 Label, 我们以一个简明的例子说明该函数.

```
surface surface(Label L, surface s, real uoffset, real voffset,
    real height=0, bool bottom=false, bool top=true);
```



```

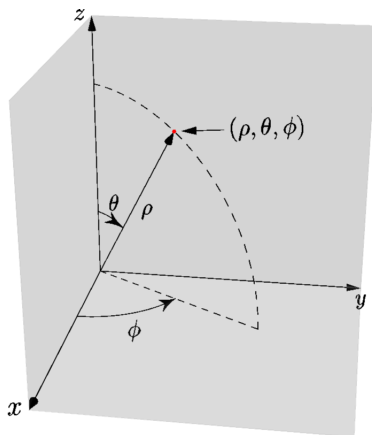
settings.tex="xelatex";
settings.prc=false;
texpreamble("\usepackage{xeCJK}");
texpreamble("\setCJKmainfont{SimSun}");
import graph3 ;
size(300);
currentprojection=orthographic(-20,-24,292);
real f(pair z){return z.x^2-z.y^2;}
surface surf=surface(f,(-6,-4),(6,4),30,30);
draw(surf,palegray,nolight,meshpen=paleblue+1pt);
string ctex="中文 \TeX";
real uoffset=0.3*30;
real voffset=0.3*30;
draw(surf.uequals(uoffset),red+dashed+1.2pt);
draw(surf.vequals(voffset),green+dashed+1.2pt);
draw(surface(scale(0.2)*ctex,surf,uoffset,voffset,height=0.2),yellow);

```

上述例子中, 曲面分成 30×30 个曲面片, 我们用 $uoffset=0.3*30$, $voffset=0.3*30$, 即 $uoffset=10$, $voffset=10$ 表示我们要标 Label 的位置, 分别用红色和蓝色虚线标出, 我们可以数数那些网线, 恰好都是第 10 条. 最后那个 `height` 表示 Label 往外突出的高度.

3.12 球面坐标

在画参数曲面时,有时用球面坐标比较方便.所谓球面坐标,即原本是用 $P = (x, y, z)$ 表示的空间上面的点,现在用 (ρ, θ, ϕ) 表示,其中 ρ 表示 P 点到原点 O 的距离, θ 表示 P 点的余纬度 (即 OP 与 z 轴所成的角,也可以看成是纬度的余角), ϕ 表示 P 点的纬度.



```
import graph3;
size(200);
currentprojection=perspective(8,2,4);
real rho=1.5;
pen bg=gray+opacity(0.2);
draw(surface((2,0,0)--(2,0,2)--(0,0,2)--(0,0,0)--cycle),bg,bg);
draw(surface((0,2,0)--(0,2,2)--(0,0,2)--(0,0,0)--cycle),bg,bg);
draw(surface((2,0,0)--(2,2,0)--(0,2,0)--(0,0,0)--cycle),bg,bg);
real theta=30;
real phi=60;
draw(arc(0,rho,0,phi,90,phi),dashed);
draw(0--rho*dir(90,phi),dashed);
draw("$\rho$",0--rho*dir(theta,phi),Arrow3);
dot(rho*dir(theta,phi),red+2);
draw("$\theta$",arc(0,rho/3,0,0,theta,phi),N+0.3E,Arrow3(HookHead2));
draw("$\phi$",arc(0,rho/2,90,0,90,phi),Arrow3(HookHead2));
arrow("$(\rho,\theta,\phi)$",rho*dir(theta,phi),20,E);
axes3("$x$","$y$","$z$",Arrow3);
```

在 Asymptote 中,与球面坐标到直角坐标的转换是如下函数

```
colatitude(triple v)
```

返回三维向量 v 与 z 轴正方向夹角 (角度制), 也就是余纬度.

```
longitude(triple v, bool warn=true)
```

返回三维向量 v 在 XY 平面的投影与 x 轴的夹角 (角度制).

```
real polar(triple v)
```

返回三维向量 v 与 z 轴正方向夹角 (弧度制).

```
azimuth(triple v)
```

返回三维向量 v 在 XY 平面的投影与 x 轴的夹角 (弧度制).

```
real dir(real colatitude, real longitude)
```

以 `colatitude` 为余纬度, 即与 z 轴正半轴的夹角. `longitude` 为经度的单位方向向量, 即方向向量在 XY 平面的投影与 x 轴的夹角.

```
real expi(real polar, real azimuth)
```

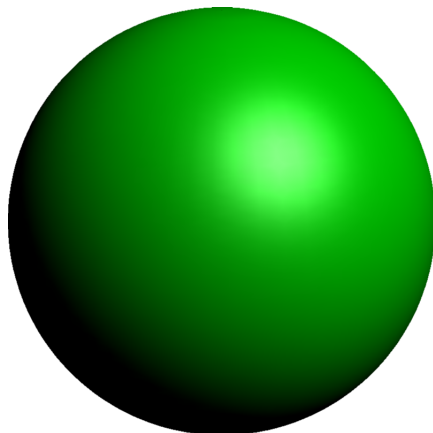
与 `dir` 类似, 但余纬度和经度分别用弧度制表示, 且分别更名为 `polar` 和 `azimuth`.

```
real length(triple v)
```

返回三维向量 v 的长度.

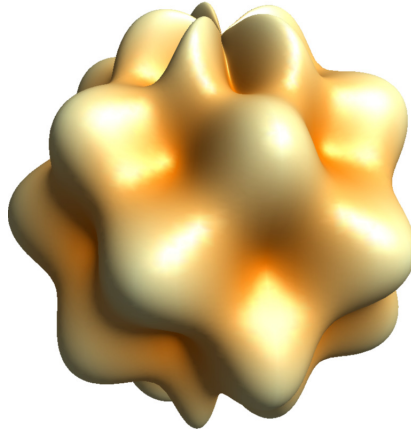
另外, `latitude(triple v)` 返回三维向量 v 与 XY 平面的的夹角 (角度制), 即纬度.

于是, 对于一个点, 如果它的球面坐标为 (ρ, θ, ϕ) , 当我们分别把 ρ, θ, ϕ 记为 `length, polar, azimuth` 以后, 那么我们可以用 `length*expi(polar, azimuth)` 来得到它的直角坐标, 这个相当与其他数学软件里面提供的球面坐标绘图.

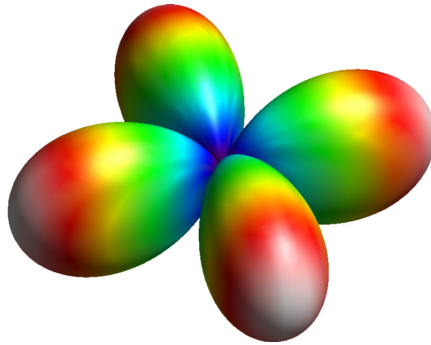


```
import graph3;
size(200);
triple f(pair z){return expi(z.x,z.y);}
surface s=surface(f,(0,0),(pi,2pi),30,Spline);
draw(s,green);
```

上面我们让半径保持为 1, 顺利成章地画出了球体, 如果我们径向函数与 θ 和 ϕ 有关, 如 $1 + \sin(5\theta) \sin(5\phi)/5$, 那么我们得到如下的效果.

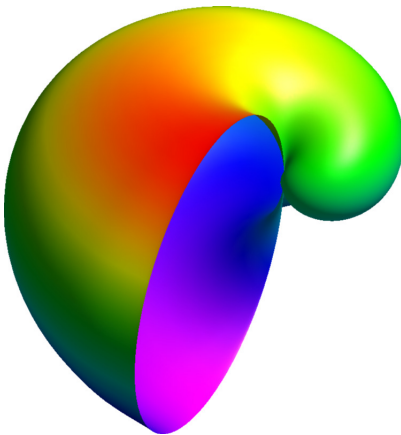


```
import graph3;
import palette;
size(200);
triple f(pair z){return (1+sin(5z.x)*sin(5z.y)/5)*expi(z.x,z.y);}
surface sf=surface(f,(0,0),(pi,2pi),40,Spline);
sf.colors(palette(sf.map(abs),Gradient(orange,paleyellow)));
draw(sf);
```



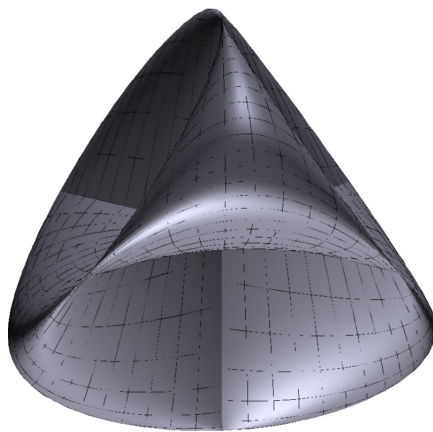
```
import graph3;
import palette;
size(200);
currentprojection=orthographic(4,2,4);
real r=2;
triple f(pair z){
    return 1/81/sqrt(2*pi)*r^2*exp(-r/3)*(sin(z.x))^2*sin(2*z.y)*expi(z.x,z.y);
}
surface s=surface(f,(0,0),(pi,2pi),50,Spline);
s.colors(palette(s.map(abs),BWRainbow()));
```

```
//draw(s);
draw(s,mean(palette(s.map(abs),BWRainbow())));
```



```
import graph3;
import palette;
size(200);
currentprojection=orthographic(4,2,4);
triple f(pair z){return z.y*sin(z.x)*expi(z.x,z.y);}
surface s=surface(f,(0,0),(pi,2*pi),40,Spline);
s.colors(palette(s.map(zpart),Rainbow()));
draw(s);
//draw(s,mean(palette(s.map(zpart),BWRainbow()))),black,nolight);
```

下面我们画出一个被称为 Steiner's Roman Surface 的曲面, 它是把单位球上面的点 (x, y, z) 变成 (yz, zx, xy) 的映射.



```
size(200);
import graph3;
```

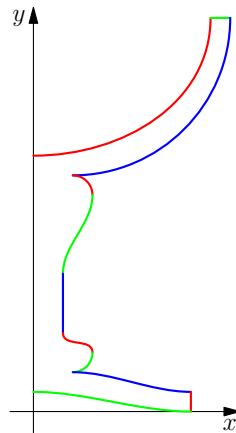
```

currentlight=White;
triple f(pair z){
    triple t=expi(z.x,z.y);
    return (t.y*t.z,t.z*t.x,t.x*t.y);
}
draw(surface(f,(0,0),(pi,2pi),50,50,Spline),meshpen=currentpen,palegrey);

```

3.13 旋转体

假定我们有一条 XY 平面的曲线, 我们可以绕着某根直线旋转生成一个旋转面, 通常选择这根直线为坐标轴, 例如 Z 轴.



```

size(200);
import graph;
path g= (0, 0.65){right}
    ..{up}(0.45, 1)
    --(0.5, 1){down}
    ..{left}(0.1,0.6)
    {right}..{down}(0.15,0.55)
    ..{down}(0.075,0.35){down}
    ..{down}(0.075, 0.2)
    ..(0.15,0.15){down}
    ..{left}(0.1,0.1){right}
    ..{right}(0.4,0.05)
    --(0.4,0){left}
    ..{left}(0,0.05);
pen[] colors={red,green,blue};
colors.cyclic=true;

```



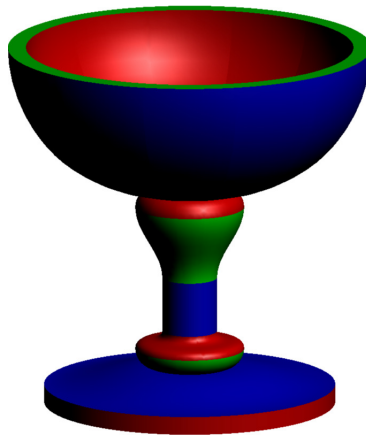
```

for(int i=0;i<length(g);++i){
    draw(subpath(g,i,i+1),colors[i]+1bp);
}
xaxis("$x$",Arrow());
yaxis("$y$",Arrow());

```

上述平面曲线由 11 段 Bezier 曲线构成, 我们依照红绿蓝的顺序循环着色, 以便看清楚它的构造.

接下去我们利用 `path3()` 函数加上 `YZplane` 参数把 `XY` 平面的曲线转换成 `YZ` 的曲线, 然后绕着 `Z` 轴旋转. 我们这里的颜色与原来的曲各段对应.



```

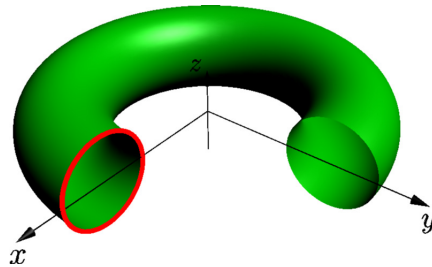
size(200);
import solids;
currentprojection=orthographic(3,3,1);
path g= (0, 0.65){right}
    ..{up}(0.45, 1)
    --(0.5, 1){down}
    ..{left}(0.1,0.6)
    {right}..{down}(0.15,0.55)
    ..{down}(0.075,0.35){down}
    ..{down}(0.075, 0.2)
    ..(0.15,0.15){down}
    ..{left}(0.1,0.1){right}
    ..{right}(0.4,0.05)
    --(0.4,0){left}
    ..{left}(0,0.05);
pen[] colors={red,green,blue};
colors.cyclic=true;
for(int i=0;i<length(g);++i){
    path3 p=path3(subpath(g,i,i+1),YZplane);

```

```

revolution R=revolution(p);
draw(surface(R),colors[i]+1bp);
}

```

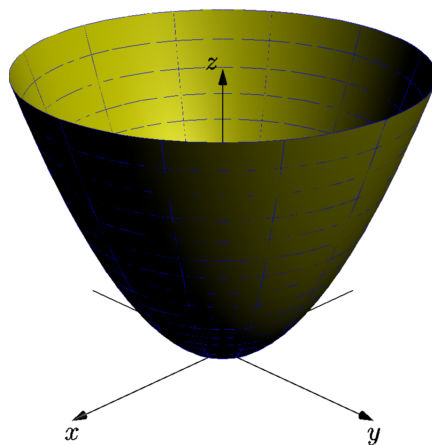


```

size(200);
import solids;
currentprojection=perspective(5,4,4);
draw(shift(3X)*Circle(0,1,Y,32),red+2);
xaxis3("$x$",0,5,Arrow3);
yaxis3("$y$",0,5,Arrow3);
zaxis3("$z$",Arrow3);
revolution torus=revolution(shift(3X)*Circle(0,1,Y,32),Z,90,360);
draw(surface(torus),green);

```

对于一个用函数定义的二维曲线, 我们很容易利用 `revolution` 画出它的绕 `Z` 轴或其他坐标轴的图形.



```

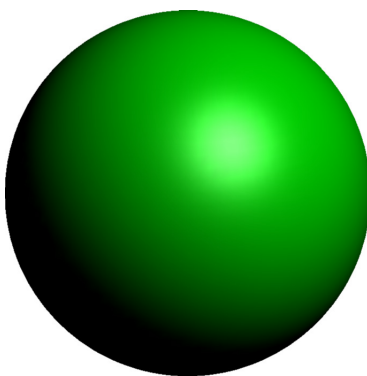
size(200,0);
import solids;
currentprojection=perspective(8,8,6);
real f(real x){return x^2;}
path p=graph(f,0,1.5,20,operator..);

```

```
//draw(p);
path3 l=path3(p,YZplane);
revolution R=revolution(l,Z);
//revolution R=revolution(f,0,1.5,20,operator..,Z);
draw(surface(R),yellow,meshpen=blue);
axes3("$x$","$y$","$z$",Arrow3());
```

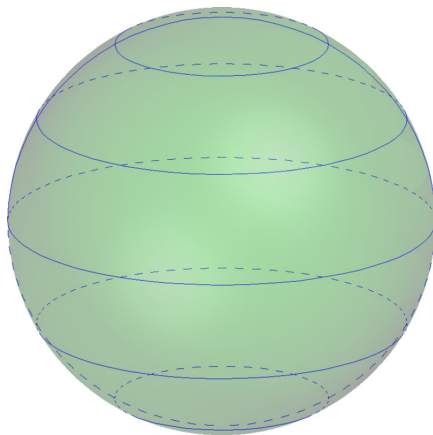
Asymptote 提供了不少常见的旋转体函数, 如 `unitsphere`, `unitcone`, `unitcylinder`, `unitsolidcone`, `unithemisphere`, `unitfrustum`(`real t1`, `real t2`) 等等.

下面以画球为例.



```
import three;
size(6cm);
currentprojection=orthographic(5,4,3);
draw(unitsphere,green);
```

提供了画球的纬线的功能, 并且做到背对观察者的纬线是虚线. 与此有关的见 `solids.asy` 里面定义的.



```
size(200);
import solids;
```

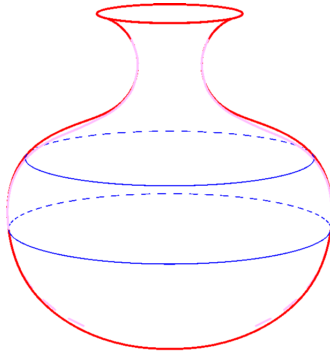
```
currentprojection=orthographic(5,4,2);
revolution sphere=sphere(1);
draw(surface(sphere),green+opacity(0.2));
draw(sphere,m=7,blue);
```

进一步的可以控制纬线的各种颜色, 详见 `solids.asy` 的代码, 这里只是列举那个函数.

```
void draw(picture pic=currentpicture, revolution r, int m=0, int n=nslice,
pen frontpen=currentpen, pen backpen=frontpen,
pen longitudinalpen=frontpen, pen longitudinalbackpen=backpen,
light light=currentlight, projection P=currentprojection)
```

有时我们希望进一步控制纬线, 在 `solids.asy` 里面提供了称为 `skeleton` 的结构.

```
struct skeleton {
    struct curve {
        path3[] front;
        path3[] back;
    }
    curve transverse;
    curve longitudinal;
}
```



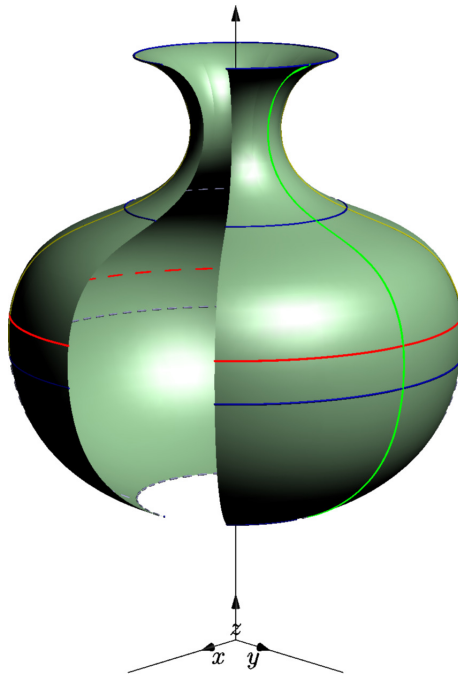
```
import solids;
size(200);
currentprojection=perspective(45,45,20);
path p=(2,3)..(3,3.5)..(4,4.5)..(4.5,6)
    ..(4,8)..(1,10)..(2,12);
path3 generator=path3(p,YZplane);
//draw(generator,pink+1bp);
```

```

revolution vase=revolution(c=(0,0,0),generator,0,360);
// 画外轮廓
draw(vase.silhouette(64),red+1bp);
// 画纬线，并且做到自动画虚线
skeleton s;
vase.transverse(s,reltime(vase.g,0.5));
vase.transverse(s,0.5*length(vase.g));
draw(s.transverse.front,blue);
draw(s.transverse.back,blue+dashed);
//画两条母线，并且做到自动画虚线
vase.longitudinal(s);
draw(s.longitudinal.front,pink+1pt);
draw(s.longitudinal.back,pink+1pt+linetype("8 8",8));

```

看一些综合的例子.



```

import solids;
size(7.5cm,0);
currentprojection=perspective(45,45,20);
draw("$x$",0--X,Arrow3);draw(0--3X);
draw("$y$",0--Y,Arrow3);draw(0--3Y);
draw("$z$",0--Z,Arrow3);draw(0--13Z,Arrow3);
path p=(2,3)..(3,3.5)..(4,4.5)..(4.5,6)

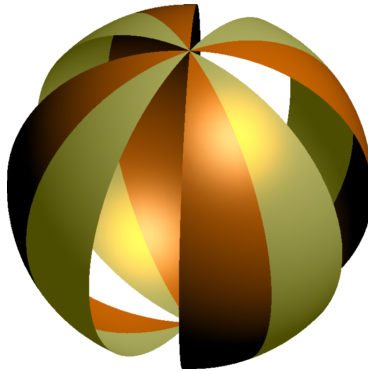
```

```

    ..(4,8)..(1,10)..(2,12);
path3 generator=path3(p,YZplane);
draw(generator,green+1bp);
revolution vase=revolution(c=(0,0,0),generator,axis=Z,-50,270);
draw(surface(vase),palegreen);
draw(vase,m=4,frontpen=blue+1bp,backpen=paleblue+1bp,
      longitudinalpen=yellow+1bp);
skeleton s;
vase.transverse(s,reltime(vase.g,0.4));
draw(s.transverse.back,red+1bp+linetype("8 8",8));
draw(s.transverse.front,red+1bp);

```

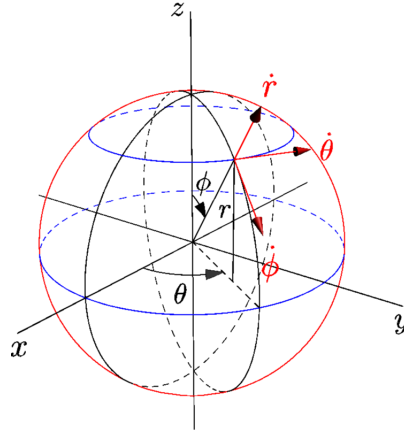
可以选择不同的画笔, 比如具有质感的 `material`, 不可见画笔 `invisible` 等. 关于 `material`, 可以参见 `three_light.asy` 的定义.



```

import solids;
size(6cm,0);
currentprojection=orthographic(1,2,2);
surface s=surface(sphere(1,n=10));
material m=material(diffusepen=grey,ambientpen=yellow,
                    emissivepen=black,specularpen=orange);
material[] p={m,invisible,orange};
p.cyclic=true;
draw(s,p);

```



```

size(200);
import solids;
currentprojection=perspective(4,3,2);
revolution Sp=sphere(1);
draw(Sp.silhouette(100),red);
triple P=dir(40,60);
draw("$r$",0--P);
draw("$\phi$",arc(0,1/3,0,0,40,60),N+0.3E,Arrow3(DefaultHead2));
draw("$\theta$",arc(0,1/2,90,0,90,60),Arrow3(DefaultHead2));
draw(0--dir(90,60),dashed);
draw(P--(xpart(P),ypart(P),0));
triple i=dir(40,60);
triple j=dir(40+90,60);
triple k=cross(i,j);
draw(Label("$\dot{r}$",1),i--i+0.5*i,red,Arrow3());
draw(Label("$\dot{\phi}$",1),i--i+0.5*j,red,Arrow3());
draw(Label("$\dot{\theta}$",1),i--i+0.5*k,red,Arrow3());
path3 pa=arc(c=0,r=1,0,60,40,60);
real l=arclength(pa);
real k=(1-l/pi);
skeleton s;
Sp.transverse(s,k*length(Sp.g));
Sp.transverse(s,reltime(Sp.g,0.5));
draw(s.transverse.front,blue);
draw(s.transverse.back,blue+dashed);
draw(arc(c=0,r=1,-10,60,150,60));
draw(arc(c=0,r=1,150,60,-10,60),dashed);

```

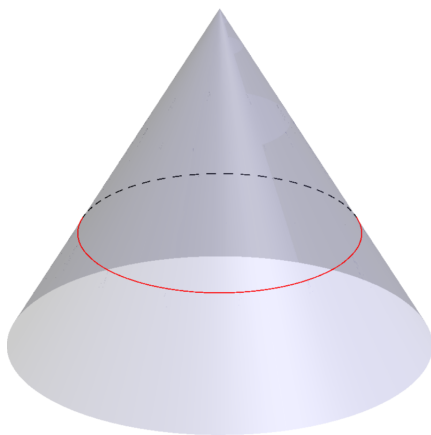
```

draw(arc(c=0,r=1,-10,0,140,0));
draw(arc(c=0,r=1,14,0,-10,0),dashed);
xaxis3("$x$",-1.5,1.5);
yaxis3("$y$",-1.5,1.5);
zaxis3("$z$",-1.5,1.5);

```

对锥体和圆柱体, 可以类似的处理那些纬线. 仅举一例如下.

利用函数生成母线, 然后画旋转体, 这方面的例子见 Asymptote 的 examples 文件夹里面的 washermethod.asy, shellmethod, sqrtx01 等例子.



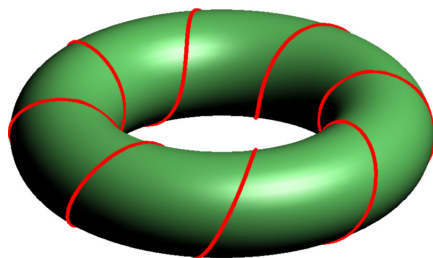
```

size(7cm,0);
import solids;
currentprojection = orthographic(50,20,25);
real r=4, h=7;
revolution Con=cone(0,r,h,axis=Z,n=4);
draw(surface(Con),lightblue+opacity(.2));
skeleton s;
Con.transverse(s,reltime(Con.g,1/3));
draw(s.transverse.back,dashed);
draw(s.transverse.front,red);

```

3.14 曲面上画曲线, 利用 lift 函数画两个曲面的交线

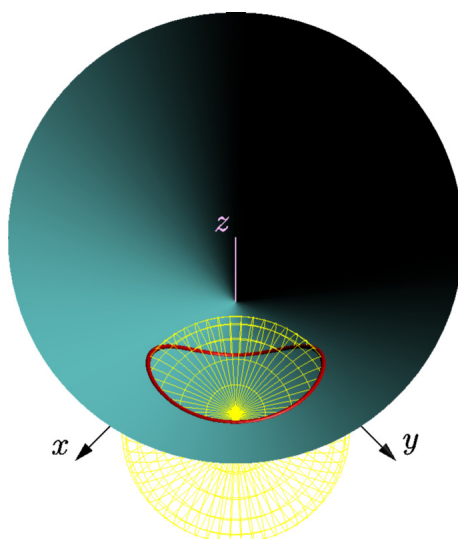
如果知道曲面上曲线的参数方程, 那么画曲面上的曲线只是分别独立的画, 前面已经提到, 利用 Asymptote 对 OpenGL 投影和 PRC 格式的支持, 独立画出来的图形是已经做到自动消隐的.



```
import graph3;
size(200,0);
currentprojection=orthographic(4,0,2);
real R=3;
real a=1;
triple f(pair z) {
    return ((a*cos(z.y)+R)*cos(z.x),(a*cos(z.y)+R)*sin(z.x),a*sin(z.y));
}
draw(surface(f,(0,0),(2pi,2pi),40,20,Spline),lightgreen);
triple c(real t){
    return ((a*cos(8*t)+R)*cos(t),(a*cos(8*t)+R)*sin(t),a*sin(8*t));
}
draw(graph(c,0,2pi,operator..),linewidth(2)+red);
```

上述曲线的单参数方程明显是与曲面的双参数方程类似, 即 $z.y$ 对应 $8*t$, $z.x$ 对应 t 因此所画的曲线是在表面上的, 其中那个 8 是让曲线绕 8 圈.

但如果要画一些不知道参数方程的曲线, 这时可以借助于 `lift` 函数.



```
// Contributed by Philippe Ivaldi , modified by cvgmt :-)
```

```

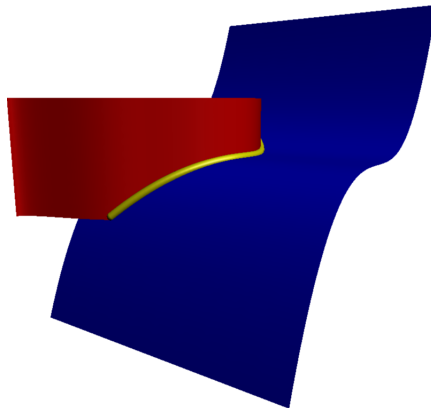
import graph3 ;
import contour;
import solids;
size (6cm,0);
currentprojection=orthographic(0.1,0.1,1) ;
real rc=1, hc=2, c=rc/hc;
draw(shift(hc*Z)*scale(rc,rc,-hc)*unitcone,lightcyan);
triple Os=(0.5,0.5,1);
real r=0.5;
draw(shift(Os)*surface(sphere(r),60),surfacepen=invisible,meshpen=yellow);
real g(pair z){return (sqrt(z.x^2+z.y^2))/c;}
//draw(surface(g,(-1,-1),(1,1),Spline),springgreen+opacity(0.2));
real f(pair z){
    real x=z.x;
    real y=z.y;
    return (x-Os.x)^2+(y-Os.y)^2+(g(z)-Os.z)^2-r^2;
}
draw(lift(g,contour(f,(-1,-1),(1,1),new real[] {0})),linewidth(2bp)+red);
xaxis3("$x$",Arrow3);
yaxis3("$y$",Arrow3);
zaxis3("$z$",p=pink);

```

所画的锥体落在由 $z = g(x, y) = \sqrt{x^2 + y^2}/c$ 确定的曲面上, 而球的方程是 $(x - Os.x)^2 + (y - Os.y)^2 + (z - Os.z)^2 - r^2 = 0$, 从这两个方程中消去 z , 得到的就是 $(x - Os.x)^2 + (y - Os.y)^2 + (g(x, y) - Os.z)^2 - r^2 = 0$, 这个正是两个曲面交线在 XY 平面的投影方程. 我们用 `contour` 生成该曲线, 然后再用 `lift` 函数把这个曲线提升到 $g(x, y)$ 上, 由此即画出两个曲面交线.

一般的, 我们可以画出 $z = g(x, y), F(x, y, z) = 0$ 确定的空间曲线.

下面用类似的方法画两个曲面的交线.



```

import graph3;
import contour;
size(200,IgnoreAspect);
currentprojection=perspective(-2,-2,2);
triple f(pair t){
    real x=t.x;
    real y=x^2;
    real z=t.y;
    return (x,y,z);
}
real g(pair t){
    real x=t.x;
    real y=t.y;
    return x^3;
}
draw(surface(f,(-2,-3),(2,3),Spline),red);
draw(surface(g,(-2,-3),(2,3),Spline),blue);
real h(pair t){
    real x=t.x;
    real y=t.y;
    return y-x^2;
}
draw(lift(g,contour(h,(-2,-2),(2,2),new real[]{0})),linewidth(4bp)+yellow);

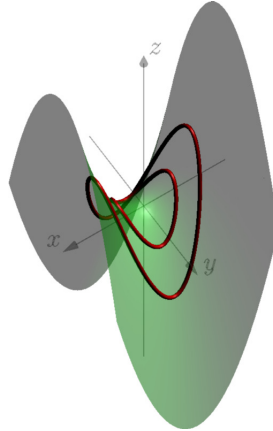
```

$y = x^2$ 要写成参数方程的形式, 此时是以 x 与 z 为参数, 于是 $y = x^2$ 写成 $y = f(x, z) = x^2$, 这就是 f 的定义.

$z = x^2$ 我们一定要写成 $z = g(x, y) = x^2$, 的形式, 否则, 后面不能画 f 与 g 的交线.

$h(x, y) = y - x^2$ 其实是要画的两个曲面在 $X - Y$ 平面的投影方程 (我们从原来两个方程中消去那个 z 就得到这个方程) `contour(h,(-2,-2),(2,2),new real[]{0})` 是我们用 `contour` 函数返回它的 `guide[]` 数据, 然后用 `lift` 函数把这些二维数据提升到函数 g 上, 从而得到 f 与 g 的交线.

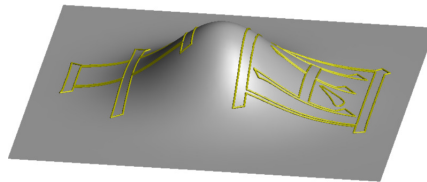
同样, 可以把二维的曲线映射到三维的曲面中.



```

import graph3 ;
import contour;
size(200);
currentprojection=orthographic(2,3,6,up=Z) ;
real g(pair z){return z.x^2-z.y^2;}
draw(surface(g,(-3,-3),(3,3),Spline),green+opacity(0.5));
real f(pair z){
    real x=z.x;
    real y=z.y;
    return x^2+y^2;
}
guide[] [] gui=contour(f,(-2,-2),(2,2),new real[] {1,3});
draw(lift(g,gui),red+linewidth(2));
axes3("$x$","$y$","$z$",opacity(0.2),Arrow3);

```



```

settings.tex="xelatex";
settings.prc=false;
texpreamble("\usepackage{xeCJK}");
texpreamble("\setCJKmainfont{SimSun}");
import graph3 ;
size(200);
currentprojection=orthographic(39.4,-264.6,121) ;

```

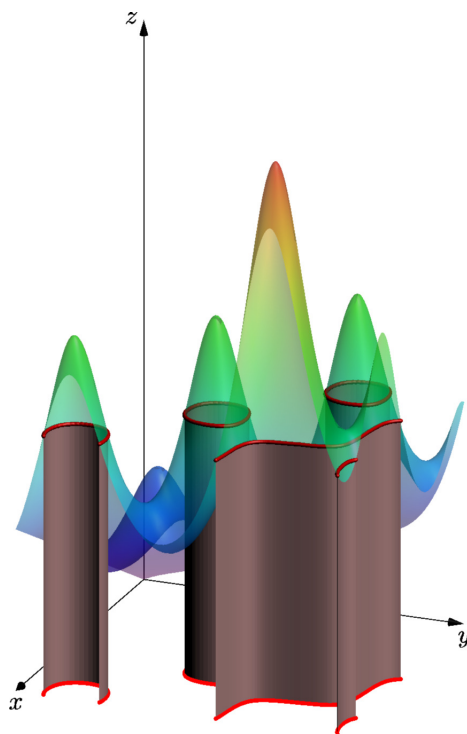
```

real f(pair z) {return 0.5+exp(-abs(z)^2);}
draw(surface(f,(-2.5,-2.5),(2.5,2.5),20,Spline),lightgray);
path[] china=scale(0.2)*texpath("中国");
guide[][] gui;
for(int i=0;i<china.length;++i){
    gui[i]=new guide[];
    int n=8;
    for(int j=0;j<n*length(china[i]);++j){
        gui[i][j]=subpath(china[i],j/n,(j+1)/n);
    }
}
draw(lift(f,gui),yellow+1pt);

```

3.15 画等高线

曲线上的等高线也是一种比较常见的图形.



```

import contour;
import palette;
import graph3;
currentprojection=orthographic(25,10,10);

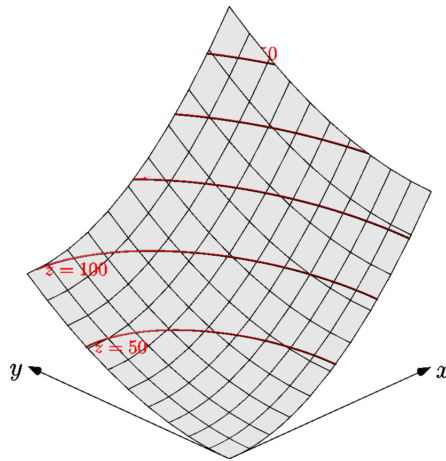
```

```

size(0,12cm);
real a=3;
real b=4;
real f(pair z) {return (z.x+z.y)/(2+cos(z.x)*sin(z.y));}
guide[] g=contour(f,(-10,-10),(10,10),new real[] {8},150);
for(guide p:g[0]){
    draw(extrude(p,8Z),palered);
    draw(path3(p),red+2pt);
}
draw(lift(f,g),red+2pt);
surface s=surface(f,(0,0),(10,10),20,Spline);
s.colors(palette(s.map(zpart),Rainbow()+opacity(0.5)));
draw(s);
axes3("$x$","$y$","$z$",Arrow3());

```

上例中我们画出了在高度为 8 的地方曲面的等高线, 也在 XY-平面的画了等高线的二维表示. `extrude` 函数的用法前面已经提及 (参见第 103 页). 另外, 为了使得 `palette` 的彩色着色不会挡住那些等高线, 我们用了 `Rainbow()+opacity(0.5)`, 即用彩色着色的同时增加了透明度.



```

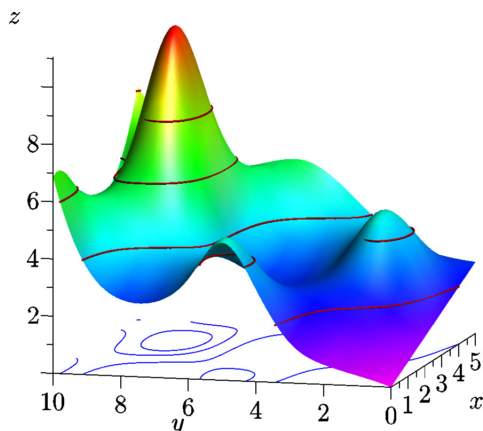
import graph3;
import contour;
size(7.5cm,0);
size3(7.5cm,IgnoreAspect);
real f(pair z) {
    return 2z.x^2-z.x+z.y^2;
}
real[] values={50,100,150,200,250};
currentprojection=orthographic(-10,-10,200);

```

```
limits((0,0,0),(10,10,300));
Label[] L=sequence(new Label(int i) {
    return scale(0.75)*Label(format("$z=%g$",values[i]),align=E,BeginPoint);
},values.length);
draw(L,lift(f,contour(f,(0,0),(10,10),values)),1bp+red);
draw(surface(f,(0,0),(10,10),nx=10,Spline),
    lightgray,meshpen=black+thick(),nolight);
xaxis3("$x$",Arrow3);
yaxis3("$y$",Arrow3);
```

我们希望标出函数 f 分别等于 50, 100, ... 250 这 5 个值的等值线. 比如其中一条就是曲面上满足 $f(x,y) = 50$ 的曲线. 我们用 `contour` 函数求出满足该方程的所以那些 XY-平面的点构成的轨迹 $\{(x,y) : f(x,y) = 50\}$, 然后再把它们提升 (`lift`) 到曲面上.

我们可以按照高度着色, 并且同时画出等高线以及等高线在 XY 平面的投影, 以便理解按照高度着色与等高线的关系.



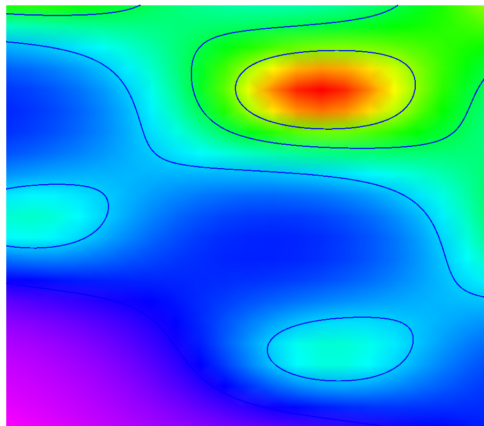
```
import graph3;
import contour;
import palette;
size(8cm,7cm,IgnoreAspect);
currentprojection=orthographic(-10,-5,4);
//limits((0,0,0),(5,10,12));
real f(pair z) {return (z.x+z.y)/(2+cos(z.x)*sin(z.y));}
real[] levels={2,4,6,8};
surface s=surface(f,(0,0),(5,10),10,Spline);
s.colors(palette(s.map(zpart),Rainbow()));
draw(s);
xaxis3(Label("$x$",position=MidPoint,align=SE),
    Bounds(Min,Min),OutTicks());
```

```

yaxis3(Label("$y$",position=MidPoint,align=SW),
        Bounds(Min,Min),OutTicks(Step=2));
zaxis3(Label("$z$",position=EndPoint,align=N+W),
        XZEquals(0,10),InTicks(beginlabel=false,endlabel=false,Label(align=Y)));
guide[] [] pl=contour(f,(0,0),(5,10),levels);
draw(lift(f,pl),1bp+red);
for(int i=0;i<pl.length;++i){
    for(int j=0;j<pl[i].length;++j){
        draw(path3(pl[i][j]),blue);
    }
}

```

我们可以把曲面的着色投影到 XY 平面, 然后从 z 轴正上方往下看, 就可以得到一个二维平面上的等高线图和所谓的密度图。



```

import graph3;
import contour;
import palette;
size(8cm,7cm,IgnoreAspect);
currentprojection=orthographic(0,0,4);
real f(pair z) {return (z.x+z.y)/(2+cos(z.x)*sin(z.y));}
real[] levels={2,4,6,8};
surface s=surface(f,(0,0),(5,10),20,Spline);
s.colors(palette(s.map(zpart),Rainbow()));
draw(planeproject(unitsquare3)*s,nolight);
guide[] [] pl=contour(f,(0,0),(5,10),levels);
for(int i=0;i<pl.length;++i){
    for(int j=0;j<pl[i].length;++j){
        draw(path3(pl[i][j]),blue);
    }
}

```

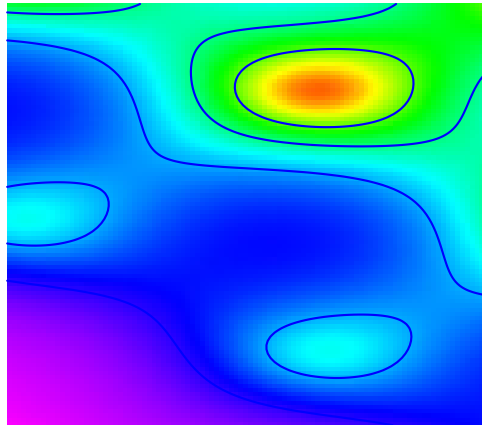


```

    }
}

```

当然, 我们也可以完全用二维作图的手段得到同样的图.

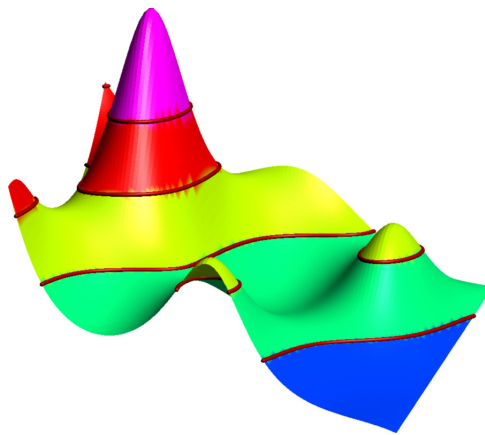


```

import graph;
import palette;
import contour;
size(8cm,7cm,IgnoreAspect);
real f(real x,real y) {return (x+y)/(2+cos(x)*sin(y));}
bounds range=image(f,Automatic,(0,0),(5,10),100,Rainbow());
real[] levels={2,4,6,8};
guide[] [] p1=contour(f,(0,0),(5,10),levels);
draw(p1,blue+1bp);

```

我们有时候希望画出颜色层次比较分明的效果, 如下图的效果.



```

import graph3;
import contour;

```

```

import palette;
size(8cm,7cm,IgnoreAspect);
currentprojection=orthographic(-10,-5,4);
//limits((0,0,0),(5,10,12));
real f(pair z) {return (z.x+z.y)/(2+cos(z.x)*sin(z.y));}
real[] levels={2,4,6,8};
surface s=surface(f,(0,0),(5,10),100);
s.colors(palette(s.map(new real(triple v) {return find(levels > v.z);}),Rainbow()));
draw(s);
draw(lift(f,contour(f,(0,0),(5,10),levels)),2bp+red);

```

其中 `find(levels > v.z)` 是一个与 `v` 有关的函数, 返回的是数组 `levels` 中超过 `v.z` 的数的指标 (index). 比如当 `v.z` 等于 5, 由于在数组 `levels` 中大于 5 的是 6, 而 6 在数组中的指标是 2, 因此 `find(levels > v.z)` 返回 2. 当然, 如果找不到那个指标, 就返回 -1. 因此, 这个与 `v` 有关的函数只能取 0,1,2,3 和 -1 这几个值, 相应的着色也就分成这 4 层 (忽略那个 -1).

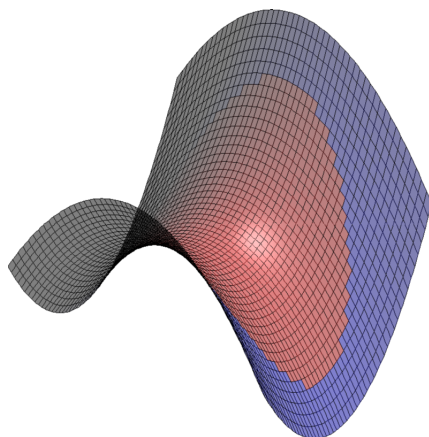
3.16 画曲面的局部

```

surface surface(real[][] f, pair a, pair b, bool[][] cond={});
surface surface(real[][] f, pair a, pair b, splinetype splinetype,
bool[][] cond={});
surface surface(real[][] f, real[] x, real[] y,
splinetype splinetype=null, bool[][] cond={})
surface surface(triple[][] f, bool[][] cond={});
surface surface(real f(pair z), pair a, pair b, int nx=nmesh, int ny=nx,
bool cond(pair z)=null);
surface surface(real f(pair z), pair a, pair b, int nx=nmesh, int ny=nx,
splinetype splinetype, bool cond(pair z)=null);
surface surface(triple f(pair z), pair a, pair b, int nu=nmesh, int nv=nu,
bool cond(pair z)=null);

```

其中的 `bool cond(pair z)` 允许我们按照这个标准去筛选出一些构成曲面的补丁, 从而画出由这些补丁构成的曲面的局部.

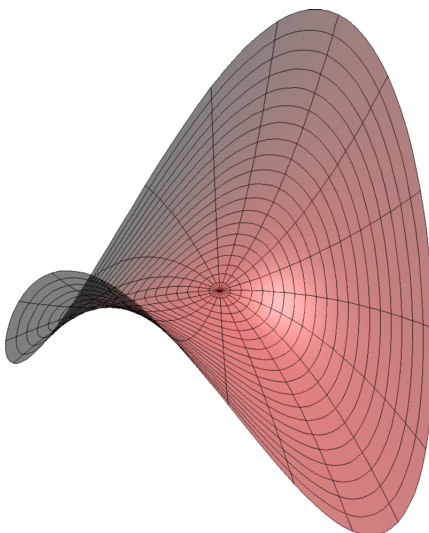


```
import graph3;
size(200);
currentprojection=orthographic(3,2,4);
real f(pair z){return z.x^2-z.y^2;}
draw(surface(f,(-1.2,-1.2),(1.2,1.2),50,Spline,
            new bool(pair z){return z.x^2+z.y^2 <=1;}),red+opacity(0.5),black);
draw(surface(f,(-1.2,-1.2),(1.2,1.2),50,Spline,
            new bool(pair z){return z.x^2+z.y^2 >1;}),blue+opacity(0.5),black);
```

这里我们用匿名函数的方法定义了 XY 平面的一个函数, 用于判断点 z 是否落在半径为 1 的圆内, 即是否满足 $z.x^2+z.y^2<1$.

当然, 我们看到画出来的效果不太好, Asymptote 正在进行这方面的工作. 在此期间, 我们采用一些技巧去完成我们的任务, 主要手法是设法把我们要画的曲面的定义域参数化.

要画圆形区域上面的曲面, 到目前为止还需要我们进行极坐标变换. 即用 $x = r \cos(\theta)$, $y = r \sin(\theta)$

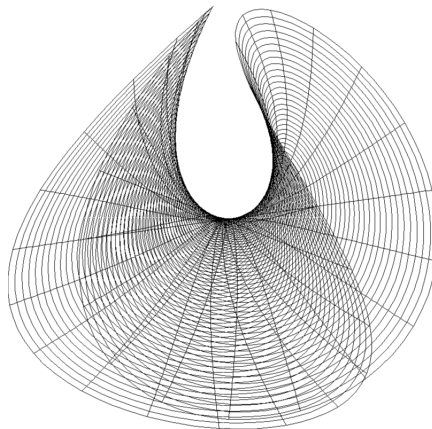


```

import graph3;
size(200,0);
currentprojection=orthographic(3,2,4);
real f(real x,real y){return x^2-y^2;}
// 对 f 进行极坐标变换;
triple g(pair t){
    real r=t.x;
    real theta=t.y;
    real x=r*cos(theta);
    real y=r*sin(theta);
    real z=f(x,y);
    return (x,y,z);
}
draw(surface(g,(0,0),(1,2pi),20,Spline),red+opacity(0.5),black);

```

类似地, 我们可以画出著名的极小曲面 Enneper-曲面, 它的定义域也是圆形区域.



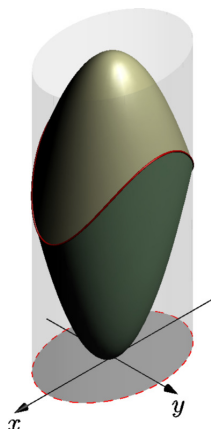
```

import graph3;
size(200,0);
currentprojection=perspective(50,-200,50);
triple f(pair t) {
    real r=t.x;
    real theta=t.y;
    pair w=(r*cos(theta),r*sin(theta));
    real x=xpart(w-w*w*w/3);
    real y=xpart((0,1)*(w+w*w*w/3));
    real z=xpart(w*w);
    return (x,y,z);
}

```

```
draw(surface(f,(0,0),(1.6,2pi),60,40,Spline),nullpen,meshpen=currentpen);
```

我们要画两个不等式 $x^2 + 3y^2 \leq z \leq 8 - x^2 - y^2$ 之间的区域, 我们采用分别把 $z = x^2 + 3y^2$ 和 $z = 8 - x^2 - y^2$ 用广义椭圆坐标化成参数方程的做法.



```
import contour;
import graph3;
size(0,200);
currentprojection=orthographic(50,50,50);
limits((-3,-2,0),(3,2,8));
real f(pair t){return 8-t.x^2-t.y^2;}
real g(pair t){return t.x^2+3t.y^2;}
real d(pair t) {
    return t.x^2/4+t.y^2/2-1;
    // return f(t)-g(t);
}
guide[] [] conto=contour(d,(-3,-3),(3,3),new real[] {0});
draw(surface(conto[0][0]..cycle),palegray);
draw(path3(conto[0][0]),red+dashed);
draw(lift(g,conto),red+1pt);
for(guide p: conto[0]){
    draw(extrude(p,8Z),palegray+opacity(0.1));
}
triple F(pair t){
    real r=t.x;
    real theta=t.y;
    real x=2*r*cos(theta);
    real y=sqrt(2)*r*sin(theta);
    real z=8-x^2-y^2;
```

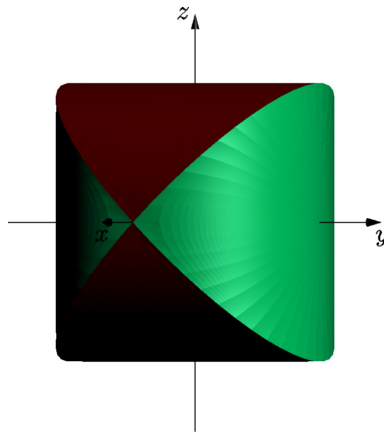
```

    return (x,y,z);
}
triple G(pair t) {
    real r=t.x;
    real theta=t.y;
    real x=2*r*cos(theta);
    real y=sqrt(2)*r*sin(theta);
    real z=x^2+3y^2;
    return (x,y,z);
}
draw(surface(F,(0,0),(1,2pi),10,Spline),paleyellow);
draw(surface(G,(0,0),(1,2pi),10,Spline),palegreen);
xaxis3("$x$",Arrow3);
yaxis3("$y$",Arrow3);

```

它们的相交线的画法就是联合两个曲面方程消去 z , 得到一个椭圆方程, 然后把该方程表示的曲线提升到曲面 g (或 f) 上.

类似的, 我们画出牟合方盖, 即两个圆柱相交所成的立体, 请对比 examples 目录里面的 `pipeintersection.asy`.



```

size(200);
import graph3;
currentprojection=orthographic(4,2,0);
real a=2;
real f(real x,real y){
    return sqrt(a^2-x^2);
}
triple g(pair t){
    real r=t.x;
    real theta=t.y;

```

```

    return (r*cos(theta),r*sin(theta),f(r*cos(theta),r*sin(theta)));
}
surface S=surface(g,(0,0),(a,2pi),50);
draw(S,brown);
draw(zscale3(-1)*S,brown);
draw(rotate(90,X)*S,springgreen);
draw(rotate(-90,X)*S,springgreen);
limits((-1.5a,-1.5a,-1.5a),(1.5a,1.5a,1.5a));
xaxis3("$x$",Arrow3());
yaxis3("$y$",Arrow3());
zaxis3("$z$",Arrow3());

```

3.17 直线段与平面的交点, 曲线与曲面的交点

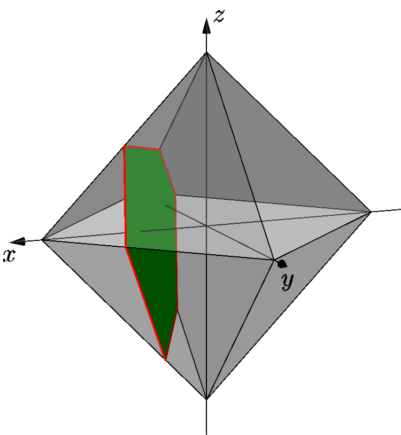
`math` 宏包提供了直线段与平面交点的 `intersect` 函数.

```

real intersect(triple P, triple Q, triple n, triple Z);

```

其中平面是由法向量 \mathbf{n} 以及一点 \mathbf{Z} 确定的. 如果只是知道平面上的三个点, 我们可以利用 `cross` 函数求出其法向量. 另外, 返回的是交点在直线段的时刻, 我们可以进一步用 `point` 函数求出该点.



```

import math;
import graph3;
size(200);
currentprojection=orthographic(2.5,6.1,1.4);
path3[] P={
    X--Y--Z--cycle,
    Y--(-X)--Z--cycle,
    (-X)--(-Y)--Z--cycle,

```

```

(-Y)--X--Z--cycle
};
for(int i=0;i<P.length;++i){
    draw(surface(P[i]),palegray+opacity(0.4));
    draw(surface(zscale3(-1)*P[i]),palegray+opacity(0.4));
    draw((P[i]));
    draw(zscale3(-1)*P[i]);
}
triple A=interp(-Y,Z,0.3);
triple B=interp(X,Z,0.5);
triple K=-Z+0.2X;
triple n=cross(A-K,B-K);
real tC=intersect(X,Y,n,K);
triple C=point(X--Y,tC);
real tD=intersect(X,-Z,n,K);
triple D=point(X--(-Z),tD);
real tE=intersect(-Y,-Z,n,K);
triple E=point((-Y)--(-Z),tE);
real tF=intersect((-Y),(-X),n,K);
triple F=point((-Y)--(-X),tF);
path3 p=A--B--C--D--E--F--cycle; draw(p,red+1pt);
draw(surface(p--cycle),green);
limits((-1.2,-1.2,-1.2),(1.2,1.2,1.2));
axes3("$x$","$y$","$z$",Arrow3());

```

上述代码先利用 $X, -X, Y, -Y, Z, -Z$ 这六个点生成正八面体. 平面由 A, B, K 这个点确定, 接下去顺次求出该平面与各条棱的交点, 由此得到平面与八面体的截面.

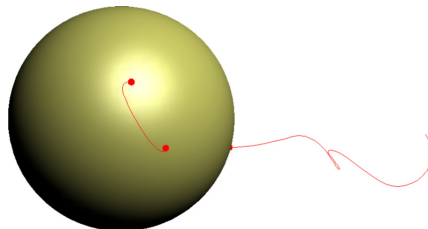
对一般的曲线和曲面, 同样的有 `intersectionpoints` 等函数.

```

real[] [] intersections(path3 p, surface s, real fuzz=-1);
triple[] intersectionpoints(path3 p, patch s, real fuzz=-1)
triple[] intersectionpoints(path3 p, surface s, real fuzz=-1);

```

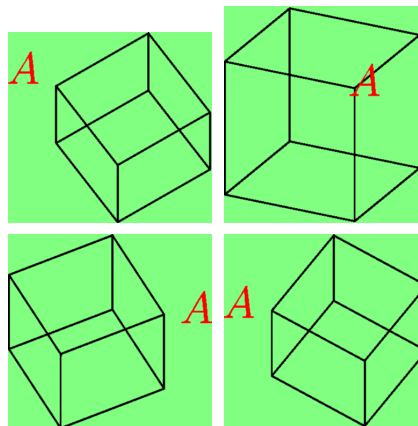
可以返回空间曲线与曲面的所有相交点.




```
//srand(seconds());
size(200);
import three;
path3 g=randompath3(15);
draw(g,red+thin());
surface s=unitsphere;
draw(s,lightyellow);
dot(intersectionpoints(g,s),red);
```

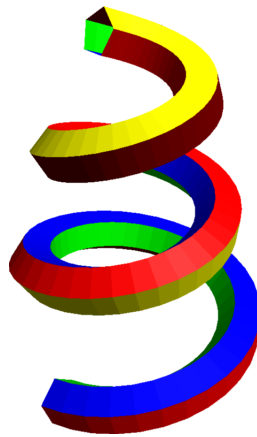
其中 `randompath3()` 函数返回一条随机空间路径. 现在的这条路径有 15 次改变方向.

3.18 多视角看同一个图形

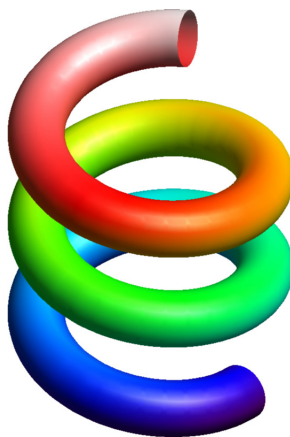


```
settings.prc=false;
import three;
currentlight.background=lightgreen;
defaultpen(fontsize(20)+linewidth(1pt));
picture pic;
size(pic,100);
draw(pic,unitbox);
label(pic,"$A$",(1,1,1),align=(1,1,1),red);
add(pic.fit(orthographic(2,1,1)),(0,0),NE);
add(pic.fit(orthographic(-2,3,5)),(0,0),SE);
add(pic.fit(orthographic(-3,2,6)),(0,0),NW);
add(pic.fit(orthographic(6,-3,8)),(0,0),SW);
```

3.19 tube 函数



```
import tube;
import graph3;
size(0,200);
currentprojection=perspective(4,3,4);
real x(real t) {return (1/sqrt(1+0.5*t^2))*cos(2*pi*t);}
real y(real t) {return (1/sqrt(1+0.5*t^2))*sin(2*pi*t);}
real z(real t) {return t;}
path3 p=graph(x,y,z,0,2.7,operator ..);
path section=scale(0.2)*polygon(5);
coloredpath cp=coloredpath(section,new pen[] {brown,green,blue,red,yellow},
                                colortype=coloredSegments);
draw(tube(p,cp));
```

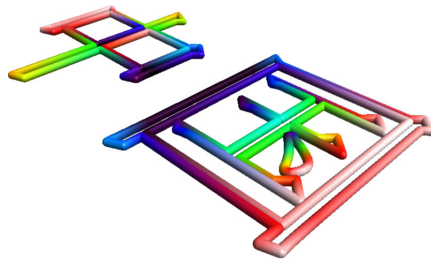


```
import tube;
import graph3;
```

```

import palette;
size(0,200);
currentprojection=orthographic(4,3,4);
triple f(real t){
    return (sin(2pi*t),cos(2pi*t),t);
}
path3 p=graph(f,0,2.7,operator ..);
//draw(tube(p,scale(0.2)*unitcircle),yellow);
tube T=tube(p,0.5);
surface surf=T.s;
surf.colors(palette(surf.map(zpart),BWRainbow()));
draw(surf);

```



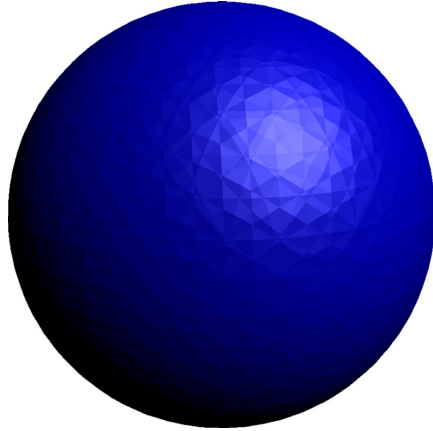
```

settings.tex="xelatex";
settings.prc=false;
texpreable("\usepackage{xeCJK}");
texpreable("\setCJKmainfont{SimSun}");
import tube;
import palette;
size(200,0);
currentprojection=perspective(0.5,-0.5,0.5,up=Z);
path[] china=texpath("中国");
path3[] p=scale3(0.5)*path3(china);
for(int i=0;i<china.length;++i){
    tube T=tube(p[i],0.2);
    surface s=T.s;
    s.colors(palette(s.map(abs),BWRainbow()));
    draw(s);
}

```

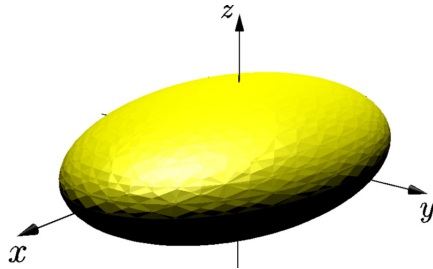
3.20 contour3 函数

有时候我们的曲面是以隐函数的形式表示的. 例如以 $(0,0,0)$ 为球心, 2 为半径的球的方程为 $x^2 + y^2 + z^2 - 4 = 0$, 这时候我们把它看成是函数 $f(x, y, z) = x^2 + y^2 + z^2 - 4$ 在 0 处的等值面.



```
import graph3;
import contour3;
size(200);
real f(real x, real y, real z){
    return x^2+y^2+z^2-4;
}
draw(surface(contour3(f,(-2.1,-2.1,-2.1),(2.1,2.1,2.1),15)),blue);
```

类似地, 我们可以画出隐函数表示的椭球面 $x^2/9 + y^2/4 + z^2 - 1 = 0$, 由于该椭球的三条半轴的长度分别是 3,2,1, 因此我们界定的范围是 $(-3,-2,-1)$ 到 $(3,2,1)$.



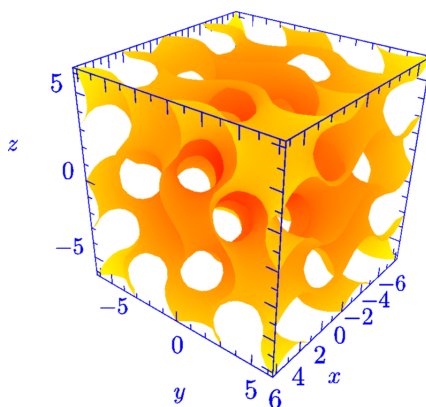
```
import graph3;
import contour3;
size(200);
currentlight=(0,0,20);
limits((-4,-3,-2),(4,3,2));
real f(real x, real y, real z){
```

```

    return x^2/9+y^2/4+z^2-1;
}
surface sf=surface(contour3(f,(-3,-2,-1),(3,2,1),12));
draw(sf,yellow);
axes3("$x$","$y$","$z$",Arrow3());

```

类似的, 我们可以画其他更复杂的隐函数表示的曲面.



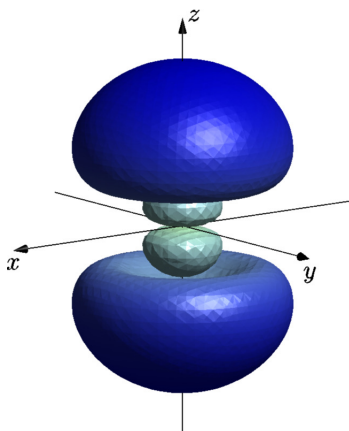
```

import graph3;
import palette;
import contour3;
size(200);
currentprojection=perspective(10,8,8);
real f(real x, real y, real z){
    return cos(x)*sin(y)+cos(y)*sin(z)+cos(z)*sin(x);
}
surface sf=surface(contour3(f,(-2pi,-2pi,-2pi),(2pi,2pi,2pi),12));
sf.colors(palette(sf.map(abs),Gradient(red,yellow)));
draw(sf,nolight);
xaxis3("$x$",Bounds(),blue,InTicks());
yaxis3("$y$",Bounds(),blue,InTicks());
zaxis3("$z$",Bounds(),blue,InTicks());

```

为了使得出来的效果好看一些, 我们采用了 `nolight` 无光照效果.

上面画的都是直角坐标系里面的隐函数方程所确定的曲面, 有时我们要画球面坐标下的等值面, 这要进行球面坐标与直角坐标的转换, 前面我们已经知道用 `expi` 函数可以把直角坐标系写成球面坐标系, 与此同时, Asymptote 也提供了 `length`, `polar`, `azimuth` 这三个函数从直角坐标系的 (x, y, z) 返回球面坐标的 (ρ, θ, ϕ) , 其中 ρ, θ, ϕ 三个的涵义如同前面的球坐标.



```

import graph3;
import contour3;
import palette;
size(200);
currentprojection=orthographic(6,8,2);
limits((-22,-22,-22),(22,22,22));
real c0=0.1;
real h(real x,real y,real z){
    triple v=(x,y,z);
    real rho=length(v);
    real theta=polar(v,warn=false);
    real phi=azimuth(v,warn=false);
    return rho*(1-rho/6)*exp(-rho/3)*cos(theta)-c0;
}
surface s=surface(contour3(h,(-20,-20,-20),(20,20,20),30));
s.colors(palette(s.map(abs),Gradient(palegreen,heavyblue)));
draw(s);
draw(zscale3(-1)*s);
axes3("$x$","$y$","$z$",Arrow3());

```

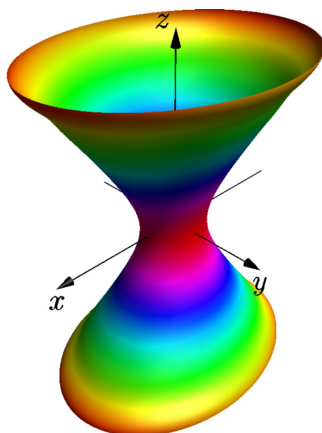
3.21 常见的二次曲面

由于到目前的 2.08 版本为止, Asymptote 还没有提供通过不等式来限制绘图范围的功能, 因此我们如果想画一些特定的图形还是比较麻烦. 这里综合前面的绘图知识画出几种常见的二次曲面.

3.21.1 单叶双曲面

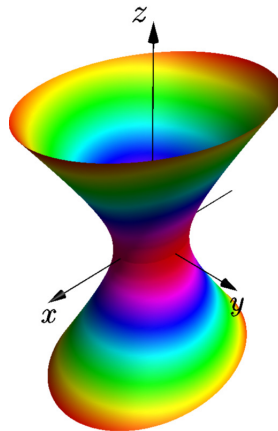
单叶双曲面是 $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$ 它的参数方程是

$$f(t, \psi) = \begin{cases} x(t, \psi) = a \cosh t \cos \psi, \\ y(t, \psi) = b \cosh t \sin \psi, \\ z(t, \psi) = c \sinh t \end{cases} \quad (3.2)$$



```
import graph3;
import palette;
size(0,200);
currentprojection=perspective(10,10,10);
limits((-10,-10,-20),(10,10,20));
real a=4;
real b=3;
real c=5;
triple f(pair t) {
    return (a*cosh(t.x)*cos(t.y), b*cosh(t.x)*sin(t.y), c*sinh(t.x));
}
surface sf=surface(f,(-2,0),(2,2pi),30,Spline);
sf.colors(palette(sf.map(abs),Wheel()));
draw(sf);
axes3("$x$", "$y$", "$z$", Arrow3());
```

我们也可以进行广义椭圆坐标变换, 即 $x=a*r*\cos(\theta)$, $y=a*r*\sin(\theta)$, 类似地解出 $z=\sqrt{c^2*(x^2/a^2+y^2-1)}$ 于是 $r \leq -1$ 或 $r \geq 1$, 同样由于对称性, 我们只是画出对应 $r \geq 1$ 的这部分, 然后用 `zscale3(-1)` 反射到 z 负方向去。



```

import graph3;
import palette;
size(0,200);
currentprojection=perspective(10,10,10);
limits((-10,-10,-20),(10,10,20));
triple f(pair t) {
    real a=4;
    real b=3;
    real c=5;
    real r=t.x;
    real theta=t.y;
    real x=a*r*cos(theta);
    real y=b*r*sin(theta);
    //real z=sqrt(c^2*(x^2/a^2+y^2/b^2-1));
    real z=sqrt(r^2-1)*c;
    return (x,y,z);
}
surface s=surface(f,(1,0),(3,2pi),50,Spline);
s.colors(palette(s.map(abs),Wheel()));

draw(surface(s,zscale3(-1)*s));

axes3("$x$","$y$","$z$",Arrow3());

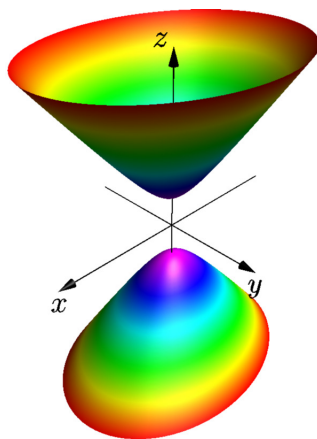
```


3.21.2 双叶双曲面

双叶双曲面是 $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = -1$, 它的参数方程是

$$f(t, \psi) = \begin{cases} x(t, \psi) = a \sinh t \cos \psi, \\ y(t, \psi) = b \sinh t \sin \psi, \\ z(t, \psi) = c \cosh t \end{cases} \quad (3.3)$$

于是可以用 `contour3` 和参数方程的画法把它画出, 这里我们还是采用广义椭圆坐标变换结合解出显式的方法的技巧.



```
import graph3;
import palette;
size(0,200);
currentprojection=perspective(10,10,10);
limits((-10,-10,-20),(10,10,20));
triple f(pair t){
    real a=4;
    real b=3;
    real c=5;
    real r=t.x;
    real theta=t.y;
    real x=a*r*cos(theta);
    real y=b*r*sin(theta);
    //real z=sqrt(c^2*(x^2/a^2+y^2/b^2+1));
    real z=sqrt(r^2+1)*c;
    return (x,y,z);
}
surface s=surface(f,(0,0),(5,2pi),40,40,Spline);
s.colors(palette(s.map(zpart),Rainbow()));
```

```
draw(surface(s,zscale3(-1)*s));
axes3("$x$","$y$","$z$",Arrow3());
```

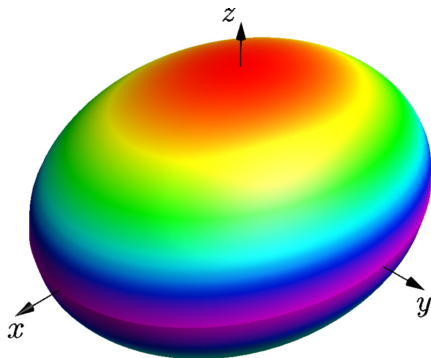
在双叶双曲面中, 当 z 固定时, 截面是一个椭圆, 因此我们进行广义椭圆坐标变换, 即 $x=a*r*\cos(\theta)$, $y=a*r*\sin(\theta)$, 然后解出 $z=\sqrt{c^2*(x^2/a^2+y^2/b^2+1)}=\sqrt{r^2+1}*c$. 由于对称性, 我们只画出对应 $r \geq 0$ 的这部分, 然后用 $zscale3(-1)$ 反射到 z 负方向去.

3.21.3 椭球面

椭球面的隐式方程是 $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$, 我们在前面已经画过. 它的参数方程是

$$f(\theta, \phi) = \begin{cases} x(\theta, \phi) = a \sin \theta \cos \phi, \\ y(\theta, \phi) = b \sin \theta \sin \phi, \\ z(\theta, \phi) = c \cos \theta \end{cases} \quad (3.4)$$

其中 $0 \leq \theta \leq \pi$, $0 \leq \phi \leq 2\pi$. 我们也可以用普通画参数方程的画法画出, 不过这里还是采用前面对双曲面用过的技巧.



```
import graph3;
import palette;
size(200,0);
currentprojection=perspective(10,10,10);
limits((-6,-5,-4),(6,5,4));
triple f(pair t){
    real a=5;
    real b=4;
    real c=3;
    real r=t.x;
    real theta=t.y;
    real x=a*r*cos(theta);
    real y=b*r*sin(theta);
    //real z=sqrt(c^2*(1-x^2/a^2+y^2/b^2));
```

```

    real z=sqrt(1-r^2)*c;
    return (x,y,z);
}
surface s=surface(f, (0,0), (1,2pi), 40, 40, Spline);
s.colors(palette(s.map(zpart), Rainbow()));
draw(surface(s, zscale3(-1)*s));
axes3("$x$", "$y$", "$z$", Arrow3());

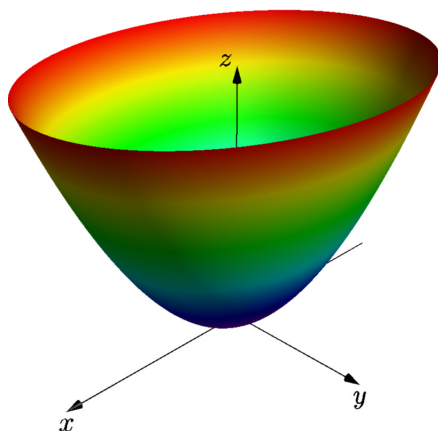
```

3.21.4 椭圆抛物面

椭圆抛物面的方程是 $z = \frac{x^2}{a^2} + \frac{y^2}{b^2}$, 也可以化成

$$f(u, v) = \begin{cases} x(u, v) = av \cos u, \\ y(u, v) = bv \sin u, \\ z(u, v) = cv^2 \end{cases} \quad (3.5)$$

的形式.



```

import graph3;
import palette;
size(0,200);
currentprojection=perspective(10,10,10);
triple f(pair t) {
    real a=4;
    real b=3;
    real c=5;
    real u=t.x;
    real v=t.y;
    return (a*v*cos(u), b*v*sin(u), c*v^2);
}

```

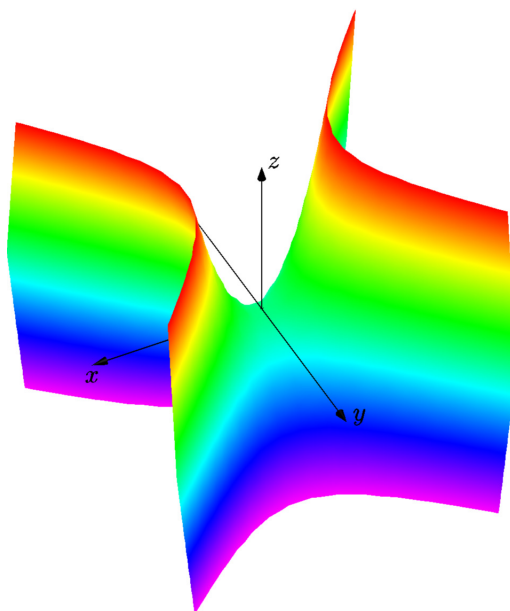
```

}
surface s=surface(f,(0,0),(2pi,1),40,40,Spline);
s.colors(palette(s.map(zpart),Rainbow()));
draw(s);
axes3("$x$","$y$","$z$",Arrow3());

```

3.21.5 双曲抛物面

双曲抛物面我们已经画过很多次了, 即 $\frac{x^2}{a^2} - \frac{y^2}{b^2} = z$. 由于我们同样希望画成被截下的效果, 因此这里采用 `contour3`, 得到的效果还是可以令人满意的.



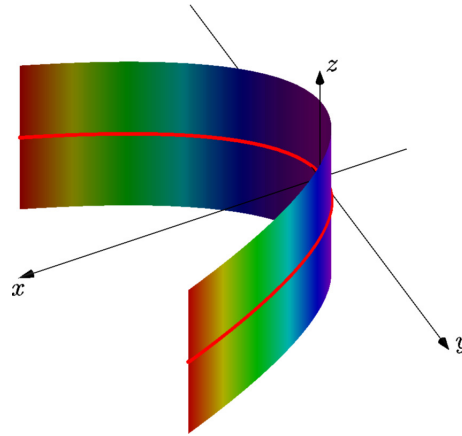
```

import graph3;
import contour3;
import palette;
real f(real x, real y, real z){
    return x^2-y^2-z;
}
size(10cm);
currentprojection=orthographic(2.5,5,5);
surface sf=surface(contour3(f,(-5,-5,-5),(5,5,5),15));
sf.colors(palette(sf.map(zpart),Rainbow()));
draw(sf,nolight);
axes3("$x$","$y$","$z$", Arrow3());

```

3.21.6 双曲柱面, 椭圆柱面, 抛物柱面

我们前面也讨论过, 这里主要是如上统一着色, 并且改为以抛物柱面为例.



```
import contour;
import palette;
import graph3;
currentprojection=orthographic(5,10,10);
limits((-4,-12,-6),(14,12,6));
size(0,200);
real p=3;
real f(real x, real y){return y^2-2p*x;}
guide[] g=contour(f,(-10,-10),(10,10),new real[] {1},50);

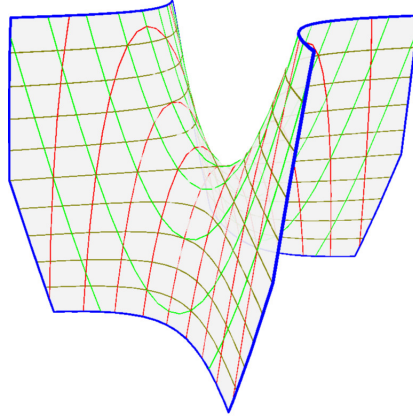
for(path p:g[0]){
    surface sf=extrude(p,4Z);
    sf.colors(palette(sf.map(xpart),Rainbow()));
    draw(surface(sf,zscale3(-1)*sf));
    draw(path3(p),red+2pt);
}

axes3("$x$","$y$","$z$",Arrow3());
```

3.22 画线框

虽然有线框和相应的画笔选项, 但有时候所给出的线框并不是我们想要的, 这时候就要手工去生成这些 WireFrame 了.

首先是 `contour3` 的线框通常都是比较杂乱, 而我们需要的是被平行于各个坐标面的平面所截出的痕迹, 这个时候就要手工用 `contour` 函数求出相应的平面曲线, 然后在移动到相应的位置.



```
import contour;
import contour3;
import three;
size(200);
currentprojection=perspective(camera=(8.7,-12.4,7.8));
real f(real x,real y,real z){
    return x^2-y^2-2z;
}
int n=10;
real a=-5;
real b=5;
for(int i=0;i<=n;++i){
    real c=interp(a,b,i/n);
    real fx(real y,real z){
        return f(c,y,z);
    }
    guide[] [] px=contour(fx,(a,a),(b,b),new real[] {0},20);
    for(int j=0;j<px[0].length;++j){
        draw(shift(c*X)*YZ()*path3(px[0][j]),((i>0 && i<n)? red : blue+1pt));
    }
}
for(int i=0;i<=n;++i){
    real c=interp(a,b,i/n);
    real fy(real x,real z){
        return f(x,c,z);
    }
}
```

```

    }
    guide[] [] py=contour(fy,(a,a),(b,b),new real[] {0},20);
    for(int j=0;j<py[0].length;++j){
        draw(shift(c*Y)*XZ()*path3(py[0][j]),((i>0 && i<n) ? green : blue+1pt));
    }
}
for(int i=0;i<=n;++i){
    real c=interp(a,b,i/n);
    real fz(real x,real y){
        return f(x,y,c);
    }
    guide[] [] pz=contour(fz,(a,a),(b,b),new real[] {0},20);
    for(int j=0;j<pz[0].length;++j){
        draw(shift(c*Z)*XY()*path3(pz[0][j]),((i>0 && i<n) ? olive : blue+1pt));
    }
}
surface sf=surface(contour3(f,(-5,-5,-5),(5,5,5),15));
draw(sf,palegray+opacity(0.9),nolight);

```

上述画法中, $f(x,y,z)$ 中令 $x=c$ 得到的 $f(c,y,z)$ 就是被 $x=c$ 这个平行于 $Y-O-Z$ 平面的平面截出的曲线的隐函数方程, 因此, 接下去就可以动用 `contour` 的功能把它画出来, 然后用对应的 `path3` 函数变为 3D 曲线再平移到相应的位置上.

用 Asymptote 的高阶函数, 我们可以把代码写得紧凑一些.

```

import contour;
import contour3;
import three;
defaultrender=render(compression=Zero,merge=true);
size(200);
currentprojection=perspective(camera=(8.7,-12.4,7.8));
int n=10;
real a=-5;
real b=5;
typedef real function(real,real);
function Tx(real f(real,real,real),real c){
    return new real(real y,real z){return f(c,y,z);};
}
function Ty(real f(real,real,real),real c){
    return new real(real x,real z){return f(x,c,z);};
}

```

```

}
function Tz(real f(real,real,real),real c){
    return new real(real x,real y){return f(x,y,c);};
}
real f(real x,real y,real z){
    return x^2-y^2-2z;
}
for(int i=0;i<=n;++i){
    real c=interp(a,b,i/n);
    guide[] [] px=contour(Tx(f,c),(a,a),(b,b),new real[] {0},20);
    guide[] [] py=contour(Ty(f,c),(a,a),(b,b),new real[] {0},20);
    guide[] [] pz=contour(Tz(f,c),(a,a),(b,b),new real[] {0},20);
    for(guide p : px[0])
        draw(shift(c*X)*YZ()*path3(p),((i>0 && i<n) ? red : blue+1pt));
    for(guide p : py[0])
        draw(shift(c*Y)*XZ()*path3(p),((i>0 && i<n) ? heavygreen : blue+1pt));
    for(guide p : pz[0])
        draw(shift(c*Z)*XY()*path3(p),((i>0 && i<n) ? olive : blue+1pt));
}
surface sf=surface(contour3(f,(a,a,a),(b,b,b),15));
draw(sf,palegray+opacity(0.9),nolight);

```

其中 `typedef real function(real,real);` 定义了一种二元函数的数据类型。

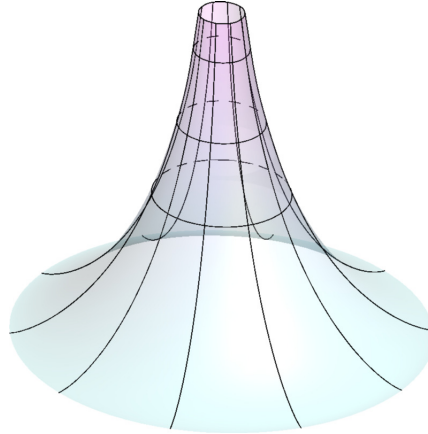
```

function Tx(real f(real,real,real),real c){
    return new real(real y,real z){return f(c,y,z);};
}

```

定义了一个三元函数 f 和实数 c 相关的函数 (泛函), 它的作用是把 c 代入 f 的第一个元中, 从而返回一个二元函数. Ty, Tz 的定义类似.

旋转体画线框更成熟一些, 前面在旋转体这一节已经讲了一些方法, 这里再举一例. 下图是一条拽物线绕着 z 轴旋转得到的伪球 (高斯曲率为负常数的常曲率曲面, 与非欧几里得几何有关)



```
// Pseudosphere:
// x = a sin(u) cos(v);
// y = a sin(u) sin(v);
// z = a (log(tan(u/2))+cos(u));
import three;
import solids;
import graph3;
import palette;
triple pseudosphere(real x) {
return (sin(x),0,cos(x)+log(tan(x/2)));
}
size(200,IgnoreAspect);
currentprojection=orthographic(160,40,100);
currentlight=(50,50,50);
path3 G=graph(pseudosphere,0.5pi,0.965pi,10,Spline);
revolution r=revolution(0,G,Z);
draw(r,1,longitudinalpen=nullpen);
surface s=surface(r);
s.colors(palette(s.map(zpart),Gradient(cyan+white+opacity(0.9),
magenta+white+opacity(0.2))));
draw(s);
skeleton s;
int n=5;
for(int i=0; i < n; ++i) {
r.transverse(s,reltime(r.g,0.5+i/n));
draw(s.transverse.back,linetype("10 10",10));
draw(s.transverse.front);
}
```

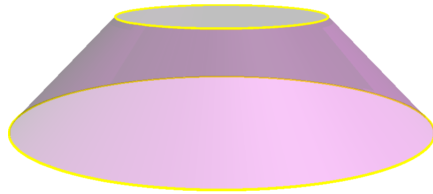
```

}
int n=10;
for(int i=0; i < n; ++i)
draw(rotate(i*360/n,0,Z)*G);

```

3.23 用 extrude 函数画直纹面

由于 `extrude` 可以画两条空间路径对应连线所成的曲面, 因此可以生成直纹面, 柱面和圆锥都是常见的直纹面, 我们先看一个简单的例子.

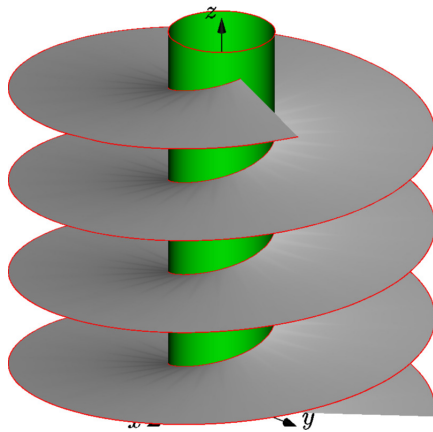


```

import three;
size(200);
path3 p=shift(Z)*unitcircle3;
path3 q=scale3(2)*unitcircle3;
draw(p^^q,yellow+1bp);
draw(extrude(p,q),magenta+opacity(0.25));

```

而螺旋面正好可以看成两条螺旋线所夹的直纹曲面, 因此我们可以如下画出.



```

// Contributed by ltongfu@bbs.ctex.org , modified by cvgmt@bbs.ctex.org :-)
import graph3;
size(200,keepAspect=false);
currentprojection=orthographic(1,1,2);

```

```

currentlight=(10,10,10);
real x(real t) {return cos(t);}
real y(real t) {return sin(t);}
real z(real t) {return t+3.1;}
path3 p=graph(x,y,z,-4,20,operator ..);
real x(real t) {return 4*cos(t);}
real y(real t) {return 4*sin(t);}
real z(real t) {return t+3.1;}
path3 q=graph(x,y,z,-4,20,operator ..);
draw(p^^q,red);
draw(extrude(p,q),palegray);
draw(path3(unitcircle)^~shift(25*Z)*path3(unitcircle),red);
draw(extrude(unitcircle,25*Z),green);
limits((0,0,0), (2,2,26));
axes3("$x$","$y$","$z$",Arrow3());

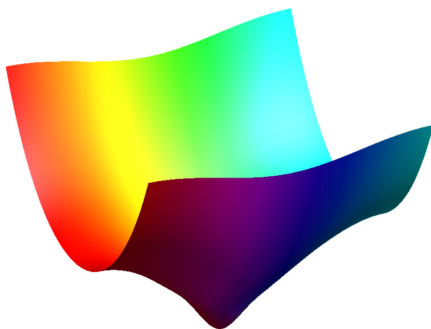
```

类似的, 我们可以画出其它直纹面, 例如 $z=x^2-y^2$ 和 $z=xy$ 和单叶双曲面等等.

3.24 画复数函数的图像

复数函数是形如 $w=f(z)$ 的函数, 其中 w 和 z 都是复数. 因此, 如果用 (z,w) 这种类似于普通函数的 (x,y) 坐标表示, 那么就实际上就需要 4 个变量, 即 $(z.x,z.y,w.x,w.y)$, 这没办法用三维的图形能够表示的. 因此, 一种做法是把 w 用它的极坐标形式, 即等同于 $\text{abs}(w)$ 和 $\text{degrees}(w)$. 然后 $(z.x,z.y,\text{abs}(w))$ 就是通常的三维曲面, 最后的用着色来表现 $\text{degrees}(w)$.

比如我们要画 $w=\sin(z)$ 这个函数, 那么我们先画 $\text{abs}(\sin(z))$ 这个函数, 然后采用随着角度变化的 `Wheel()` 函数着色.



```

import graph3;
import palette;
size(200);
currentprojection=orthographic(1,2,1);

```

```

surface s=surface(new real(pair z) {return abs(sin(z));},
                  (-pi/2,-pi/2),(pi/2,pi/2),10,Spline);

s.colors(palette(s.map(new real(triple v){
    return degrees(sin((v.x,v.y)),warn=false);}),Wheel()));
draw(s);

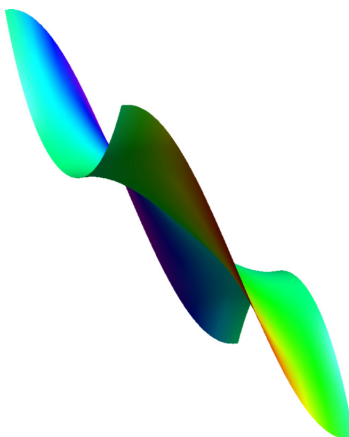
```

这里我们采用了匿名函数的办法定义一个这种与 `triple v` 相关的函数 `degrees(sin((v.x,v.y)),warn=false)`, 使得这个着色函数反映的是 $\sin(z)$ 幅角的变化.

类似地, 我们可以画其他更为复杂的复数函数, 比如 $\gamma(z)$ 函数, 具体见 `gamma3.asy`.

当然, 我们也可以画 $(z.x, z.y, w.x)$ 的图像, 即画出复数函数实部的图形, 然后用着色表现复数函数的虚部 $w.y$.

下面仅举一例, 画出 $f(z)=\sin(z)$ 的图像, 以便与前面的利用模和幅角的画法对比. 当然, 也可以换成其他的复数函数.

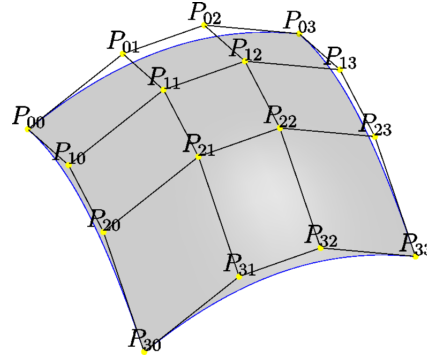


```

import graph3;
import palette;
size(200);
pair f(pair z){return sin(z);}
currentprojection=orthographic(1,2,1);
surface s=surface(new real(pair z) {return f(z).x;},
                  (-pi/2,-pi/2),(pi/2,pi/2),10,Spline);
s.colors(palette(s.map(new real(triple v){
    return f((v.x,v.y)).y;}),Rainbow()));
draw(s);

```

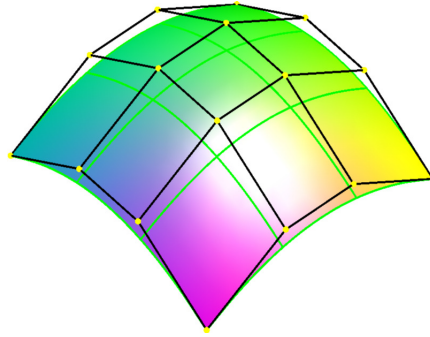
3.25 Bezier 曲面的构造



```

import graph3;
size(200);
currentprojection=orthographic(3.5,-1.5,5.5);
triple[] [] P={
    {(-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5)},
    {(-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0)},
    {(0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0)},
    {(1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5)}
};
patch s=patch(P);
draw(s.external(),blue);
draw(surface(s),opacity(0.2));
for(int i=0;i<P.length;++i){
    for(int j=0;j<P[i].length;++j){
        label(Label(format("$P_{%d}",i)+format("%d}$",j)),align=Z,P[i][j]);
        dot(P[i][j],yellow);
    }
}
for(int i=0;i<P.length;++i){
    draw(operator--(...P[i]));
}
P=transpose(P);
for(int i=0;i<P.length;++i){
    draw(operator--(...P[i]));
}

```



```

import graph3;
import palette;
size(200);
currentprojection=orthographic(1.3,1.5,2.4);
currentlight=(-2,2,10);
triple[] [] [] P={
    {
        {(-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5)},
        {(-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0)},
        {(0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0)},
        {(1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5)}
    }
};
surface sf=surface(P);
//draw(sf,yellow);
for(int i=0;i<P.length;++i){
    for(int j=0;j<P[i].length;++j){
        for(int k=0;k<P[i][j].length;++k){
            dot(P[i][j][k],linewidth(3)+yellow);
        }
    }
}
for(int i=0;i<P.length;++i){
    for(int j=0;j<P[i].length;++j){
        for(int k=0;k<P[i][j].length-1;++k)
            draw(P[i][j][k]--P[i][j][k+1],linewidth(1));
    }
}
for(int i=0;i<P.length;++i){

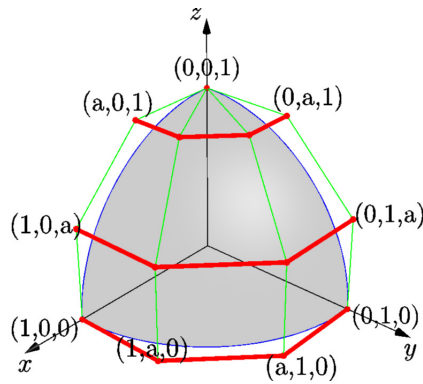
```

```

for(int j=0;j<P[i].length-1;++j){
    for(int k=0;k<P[i][j].length;++k)
        draw(P[i][j][k]--P[i][j+1][k],linewidth(1));
}
}
for(int i=0; i <4; ++i) {
    draw(sf.s[0].uequals(i/3),linewidth(1)+green);
    draw(sf.s[0].vequals(i/3),linewidth(1)+green);
}
draw(surface(sf.s[0].external(),sf.s[0].internal(),new pen[]{green,cyan,magenta,yellow}));
//sf.colors(palette(sf.cornermap(ypart),Rainbow()));
//draw(sf);

```

利用 Bezier 退化四边形构造球面三角形.



```

import graph3;
size(200);
currentprojection=perspective(4.4,3.9,3.4);
limits((0,0,0),(1.4,1.4,1.4));
real a=4/3*(sqrt(2)-1);
triple[] p={
    {(1,0,0),(1,0,a),(a,0,1),(0,0,1)},
    {(1,a,0),(1,a,a),(a,a^2,1),(0,0,1)},
    {(a,1,0),(a,1,a),(a^2,a,1),(0,0,1)},
    {(0,1,0),(0,1,a),(0,a,1),(0,0,1)}
};
path3 g=p[0][0]{Z}..{-X}p[0][3]..p[3][3]{Y}..{-Z}p[3][0]{X}..{-Y}cycle;
draw(g,blue);
draw(surface(g,new triple[]{p[1][1],p[1][2],p[2][2],p[2][1]}),opacity(0.2));
for(int i=0;i<p.length;++i){

```

```

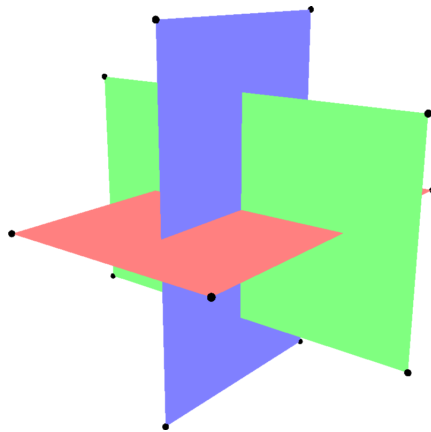
draw(operator--(...p[i]),green);
dot(p[i],red);
}
triple[] [] q=transpose(p);
for(int i=0;i<q.length;++i){
    draw(operator--(...q[i]),red+2pt);
}
label("(1,0,0)",p[0][0],p[0][0]);
label("(1,0,a)",p[0][1],p[0][1]);
label("(a,0,1)",p[0][2],p[0][2]);
label("(1,a,0)",p[1][0],p[1][2]);
label("(a,1,0)",p[2][0],p[2][0]);
label("(0,1,0)",p[3][0],p[3][0]);
label("(0,1,a)",p[3][1],p[3][1]);
label("(0,a,1)",p[3][2],p[3][2]);
label("(0,0,1)",p[3][3],p[3][3]);
axes3("$x$","$y$","$z$",Arrow3());

```

3.26 各种多面体

Jens Schwaiger 写的同时适用于 `render=0` 和真 3D 的 `polyhedron_js.asy` 宏包, 里面大部分内容是精心构造好的各种多面体的数据. 可以直接运用. 当然, 某些时候我们为了我们的需求, 还是要手工构造. 我们看一个正 20 面体的构造过程.

先构造出四个躺在坐标平面的矩形, 矩形的 12 个顶点就是正 20 面体的顶点.



```

import graph3;
size(200);
real c=(1+sqrt(5))/2;

```

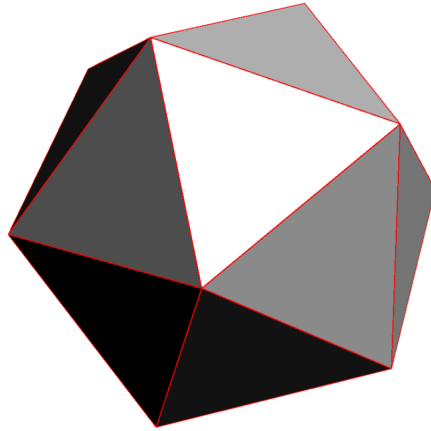


```

triple[] z={(c,1,0),(-c,1,0),(-c,-1,0),(c,-1,0)};
triple[] x={(0,c,1),(0,-c,1),(0,-c,-1),(0,c,-1)};
triple[] y={(1,0,c),(1,0,-c),(-1,0,-c),(-1,0,c)};
draw(surface(z[0]--z[1]--z[2]--z[3]--cycle),lightred,nolight);
draw(surface(x[0]--x[1]--x[2]--x[3]--cycle),lightgreen,nolight);
draw(surface(y[0]--y[1]--y[2]--y[3]--cycle),lightblue,nolight);
dot(z);
dot(x);
dot(y);

```

然后列出这 12 个顶点的每一个顶点所对的 5 个点. 另外, 为了使得所有的顶点都在单位球上, 我们还进行了放缩.



```

import graph3;
size(200);
real c=(1+sqrt(5))/2;
triple[] z={(c,1,0),(-c,1,0),(-c,-1,0),(c,-1,0)};
triple[] x={(0,c,1),(0,-c,1),(0,-c,-1),(0,c,-1)};
triple[] y={(1,0,c),(1,0,-c),(-1,0,-c),(-1,0,c)};
triple[] [] Q={
    {(c,1,0),(1,0,-c),(0,c,-1),(0,c,1),(1,0,c),(c,-1,0)},
    {(-c,1,0),(0,c,1),(0,c,-1),(-1,0,-c),(-c,-1,0),(-1,0,c)},
    {(-c,-1,0),(-c,1,0),(-1,0,-c),(0,-c,-1),(0,-c,1),(-1,0,c)},
    {(c,-1,0),(c,1,0),(1,0,c),(0,-c,1),(0,-c,-1),(1,0,-c)},
    {(0,c,1),(0,c,-1),(-c,1,0),(-1,0,c),(1,0,c),(c,1,0)},
    {(0,-c,1),(0,-c,-1),(-c,-1,0),(-1,0,c),(1,0,c),(c,-1,0)},
    {(0,-c,-1),(0,-c,1),(c,-1,0),(1,0,-c),(-1,0,-c),(-c,-1,0)},
    {(0,c,-1),(0,c,1),(c,1,0),(1,0,-c),(-1,0,-c),(-c,1,0)},
    {(1,0,c),(-1,0,c),(0,-c,1),(c,-1,0),(c,1,0),(0,c,1)},
    {(-1,0,-c),(1,0,-c),(0,-c,-1),(-c,-1,0),(-c,1,0),(0,c,-1)},
    {(1,0,c),(-1,0,c),(0,-c,1),(c,-1,0),(c,1,0),(0,c,1)},
    {(-1,0,-c),(1,0,-c),(0,-c,-1),(-c,-1,0),(-c,1,0),(0,c,-1)}
};

```

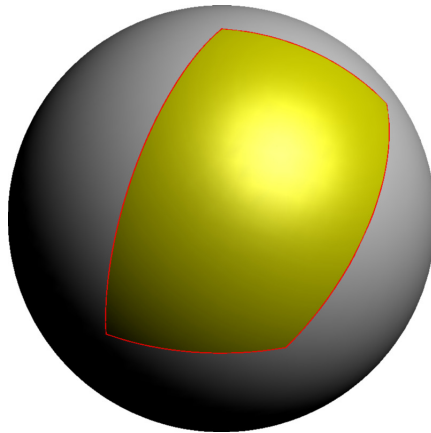
```

    {(1,0,-c),(-1,0,-c),(0,-c,-1),(c,-1,0),(c,1,0),(0,c,-1)},
    {(-1,0,-c),(1,0,-c),(0,c,-1),(-c,1,0),(-c,-1,0),(0,-c,-1)},
    {(-1,0,c),(1,0,c),(0,c,1),(-c,1,0),(-c,-1,0),(0,-c,1)}
};
real R=abs(Q[0][0]);
triple[] [] P;
for(int i=0; i < Q.length; ++i){
    P[i]=new triple[] ;
    for(int j=0; j < Q[i].length; ++j){
        P[i][j]=Q[i][j]/R;
    }
}
for(int i=0;i<P.length;++i){
    for(int j=1;j<P[i].length;++j){
        draw(P[i][0]--P[i][j],red);
        draw(surface(P[i][0]--P[i][j]--P[i][j%5+1]--cycle),palegray);
    }
}
}

```

3.27 球面多边形, 足球与 C60

下面我们先讨论球面四边形以及球面三角形的画法, 更进一步, 我们可以画出足球. 先看球面四边形. 我们可以把球面多边形参数化, 方法是对满足 $0 < x < 1, 0 < y < 1$ 的点 (x, y) , 先找到相对于球面多边形一组对边比例都是 x 的两个点, 然后利用这两个点生成一条球面曲线, 接着在这条球面曲线上取比例为 y 的点. 这样一来, 就把球面四边形变成是定义在 $(x, y) : 0 < x < 1, 0 < y < 1$ 上的参数曲面.



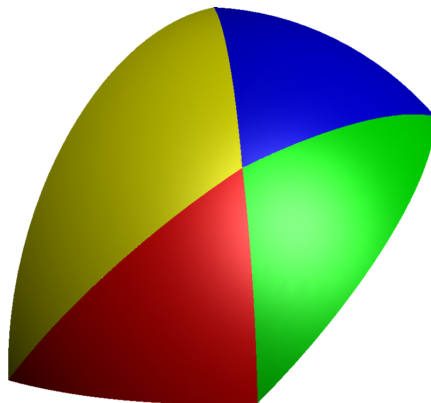
```
size(200);
```

```

import graph3;
currentprojection=perspective(5.5,3.1,4.3);
surface Surf(triple center, triple A, triple B, triple C, triple D,
             int nu=3, int nv=nu) {
    path3 p1=arc(center,A,B);
    path3 p2=arc(center,B,C);
    path3 p3=arc(center,C,D);
    path3 p4=arc(center,D,A);
    triple surf(pair t) {
        real u=t.x;
        real v=t.y;
        path3 cr=arc(center,relpoint(p1,u),relpoint(reverse(p3),u));
        return relpoint(cr,v);
    };
    return surface(surf,(0,0),(1,1),nu,nv,Spline);
}
real r=1+0.005;
triple A=r*unit((1,0,0));
triple B=r*unit((1,1,0));
triple C=r*unit((-0.2,0.5,0.5));
triple D=r*unit((0,0,1));
draw(Surf(0,A,B,C,D,4,4),yellow);
draw(unitsphere,palegray);
draw(arc(0,A,B)^~arc(0,B,C)^~arc(0,C,D)^~arc(0,D,A),red+thick());

```

把球面四边形的做法修改一下, 即把相邻的两个点充分靠近, 就得到球面三角形了.



```

size(200);
import graph3;

```

```

surface sphericaltriangle(triple center, triple A, triple B, triple C,
                        int nu=8, int nv=nu) {
    path3 tri1=arc(center,A,B);
    path3 tri2=arc(center,A,C);
    path3 tri3=arc(center,B,C);
    triple tri(pair p) {
        path3 cr=arc(0,relpoint(tri2,p.x),relpoint(tri3,p.x));
        return relpoint(cr,p.y);
    };
    return surface(tri,(0,0),(1-sqrtEpsilon,1),nu,nv,Spline);
}

triple A=unit((1,0,0));
triple B=unit((1,1,0));
triple C=unit((-0.2,0.5,0.5));
triple D=unit((0,0,1));
triple M=unit((A+B+C+D)/4);
draw(sphericaltriangle(0,A,B,M),red);
draw(sphericaltriangle(0,B,C,M),green);
draw(sphericaltriangle(0,C,D,M),blue);
draw(sphericaltriangle(0,D,A,M),yellow);

```

更进一步, 利用正 20 面体, 我们得到包括 60 顶点的多面体, 利用上述球面三角形就可以画出足球了. 代码请参见 examples 目录中由笔者提供的 soccerball.asy.



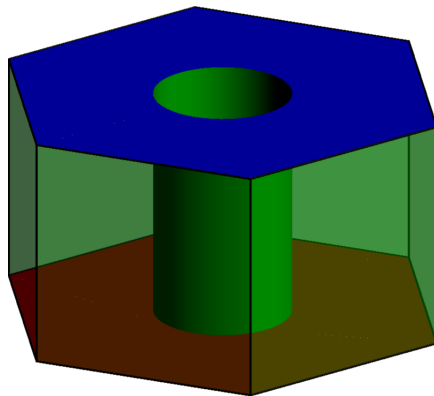
2009 年 6 月份, Asymptote 提供了画有洞的空间平面图形的功能, 主要是 `planar=true` 选项, 我们先看两个例子.



```

import three;
size(200);
path p=unitcircle;
path q=scale(2)*p;
draw(surface(reverse(path3(p))^^path3(q),planar=true),yellow);

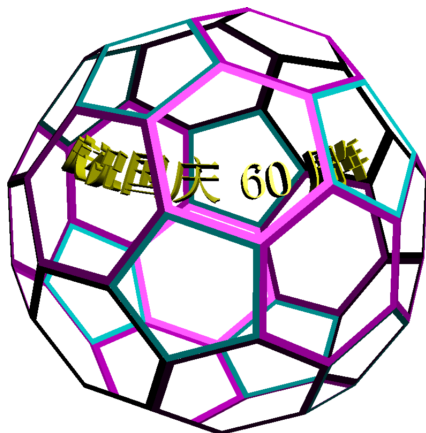
```



```

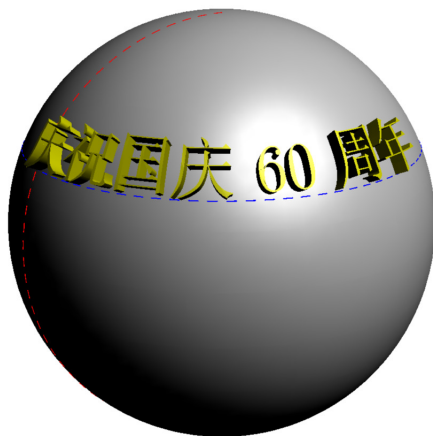
size(200);
import three;
currentprojection=orthographic(3,4,2);
path p=reverse(polygon(6));
path q=scale(0.3)*unitcircle;
path3 pp=path3(p);
path3 qq=path3(q);
draw(surface(pp^^qq,planar=true),red);
draw(surface(shift(Z)*pp^^qq,planar=true),blue);
for(path h : p)
    draw(extrude(h,Z),green+opacity(0.5),black+1pt);
for(path h : q)
    draw(extrude(h,Z),green);

```



利用 Asymptote 对空间平面图实现的镂空功能, 笔者画了一副庆祝国庆 60 周年的图画, 且已经加入到 examples 目录里面成了 truncatedIcosahedron.asy. 当然, 里面没有中文, 用前面的把字体提升到 3D 曲面的做法就行了. 即加入

```
settings.tex="xelatex";
settings.prc=false;
texpreamble("\usepackage{xeCJK}");
texpreamble("\setCJKmainfont{SimSun}");
import solids;
viewportmargin=(10,10);
size(200);
revolution sphere=sphere(0.75);
surface s=surface(sphere);
string C60="庆祝国庆周年60";
real uoffset=-0.8;
real voffset=7.5;
real h=0.05;
draw(surface(xscale(0.35)*yscale(0.08)*C60,s,uoffset,voffset,h),yellow);
draw(s.uequals(uoffset),red+dashed);
draw(s.vequals(voffset),blue+dashed);
draw(surface(s),palegray);
```

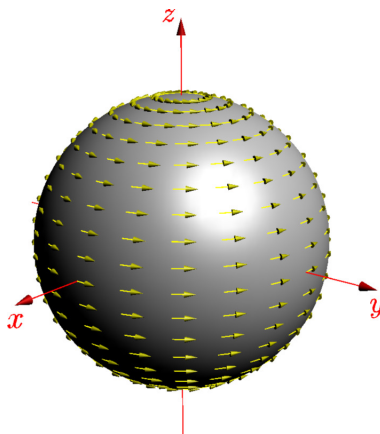


3.28 曲面上的向量场

在 graph3 里面定义了 `vectorfield` 函数可以画曲面向量场.

```
picture vectorfield(path3 vector(pair v), triple f(pair z), pair a, pair b,
int nu=nmesh, int nv=nu, bool truesize=false,
real maxlength=truesize ? 0 : maxlength(f,a,b,nu,nv),
bool cond(pair z)=null, pen p=currentpen,
arrowbar3 arrow=Arrow3, margin3 margin=PenMargin3)
```

下面画出的是由球面的经线的切向量所构成的球面上的向量场.



```
import graph3;
size(200);
triple f(pair z){return expi(z.x,z.y);}
path3 vector(pair z) {
    real t=sqrtEpsilon;
```

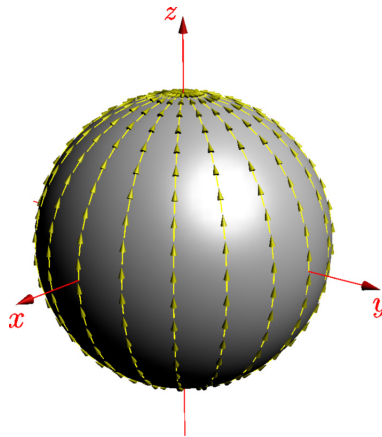
```

triple U=(expi(z.x,z.y+t)-expi(z.x,z.y))/t;
U=unit(U);
return 0--U;
}
add(vectorfield(vector,f,(0,0),(pi,2pi),20,0.15,yellow));
draw(unitsphere,palegray);
limits((-1.5,-1.5,-1.5),(1.5,1.5,1.5));
axes3("$x$","$y$","$z$",red,Arrow3());

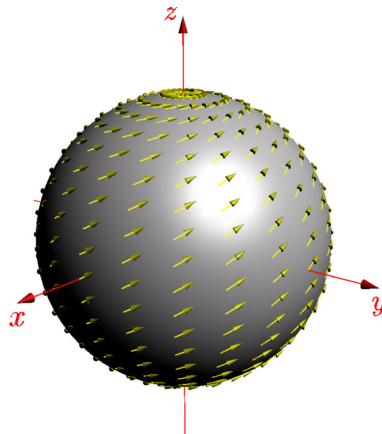
```

我们用 $(\text{expi}(z.x+t,z.y)-\text{expi}(z.x,z.y))/t$; 求出经线的切向量.

如果改用 $(\text{expi}(z.x,z.y+t)-\text{expi}(z.x,z.y))/t$; 就是纬线的切向量.



也可以有一定斜驶的效果 (与纬线有一定的夹角, 这里是 $\pi/6$)



```

import graph3;
size(200);
triple f(pair z){return expi(z.x,z.y);}
path3 vector(pair z) {
    real t=sqrtEpsilon;

```



```

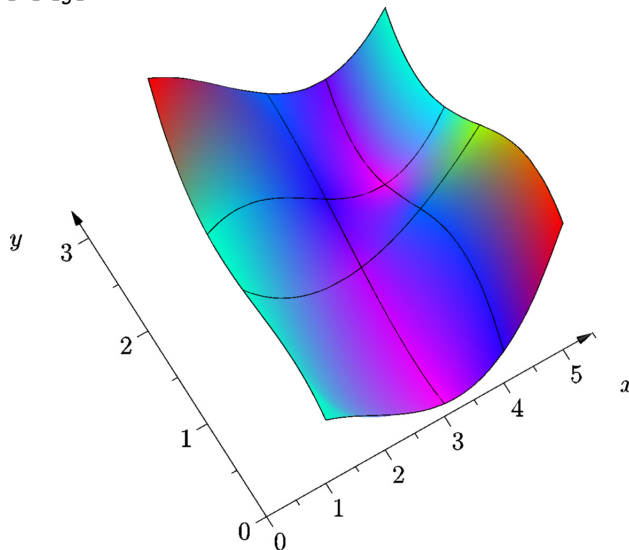
triple U=(expi(z.x,z.y+t)-expi(z.x,z.y))/t;
triple V=(expi(z.x,z.y)-expi(z.x+t,z.y))/t;
triple K=cos(pi/6)*unit(U)+sin(pi/6)*unit(V);
K=unit(K);
return 0--K;
}
add(vectorfield(vector,f,(0,0),(pi,2pi),20,0.15,yellow));
draw(unitsphere,palegray);
limits((-1.5,-1.5,-1.5),(1.5,1.5,1.5));
axes3("$x$","$y$","$z$",red,Arrow3());

```

3.29 离散数据的可视化

| | | | |
|-----|-----|-----|-----|
| 1 | 3 | 4 | 5 |
| 0 | 1.4 | 2 | 3 |
| 1.0 | 1.0 | 1.0 | 2.0 |
| 0.2 | 0.3 | 0.5 | 0.7 |
| 0.5 | 0.7 | 0.2 | 0.4 |
| 2 | 1.5 | 0.9 | 1.0 |

假定我们有如上一组数据, 其中的第一行是所有横坐标, 由它可以构成一维数组 $x[]$, 它的长度为 4, 第二行是所有纵坐标, 由它可以构成一维数组 $y[]$, 它的长度也是 4. 接下去的 4×4 的二维数组, 对应 $(x[i], y[j])$ 上的高度 $z[i][j]$.



```

import palette;
import graph3;

```

```

size3(200,IgnoreAspect);
currentprojection=orthographic(-2,-2,4);
file in=input("data-1.txt").line();
real[] x=in;
real[] y=in;
real[] [] z=in.dimension(0,0);
surface s=surface(z,x,y,Spline);
s.colors(palette(s.map(zpart), Rainbow()));
draw(s,nolight,meshpen=thick());
limits((0,0,0),(1.1*max(x),1.1*max(y),1.1*max(z)));
xaxis3("$x$",OutTicks(),Arrow3(),above=true);
yaxis3("$y$",OutTicks(),Arrow3(),above=true);

```

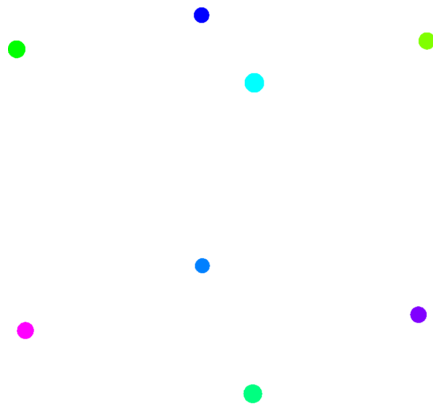
我们用 `file in=input("data.txt").line();` 逐行读入数据文件 `data.txt`, 然后让一维数组 `x` 接受了文件 `in` 的第一行, 之后的一维数组 `y` 就接受了第二行, 剩下的用 `in.dimension(0,0);` 把二维数组读入到 `real[] [] z` 中.

观察数组, `z[0][3]` 与 `z[3][0]` 处的数值最大, 相应地在 $(x[0], y[3])=(1, 3)$ 和 $(x[3], y[0])=(5, 0)$ 处是光谱中的红色.

```

1 0 0
0 1 0
0 0 1
0 0 0
1 1 1
1 1 0
1 0 1
0 1 1

```



```

import three;

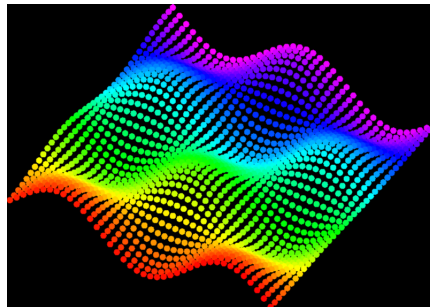
```

```

import palette;
size(200);
file f=input("scatter.txt");
triple[] a;
while(true) {
    real x=f;
    real y=f;
    real z=f;
    if(eof(f)) break;
    a.push((x,y,z));
}
for(int i=0; i < a.length; ++i)
    dot(a[i],Rainbow(a.length)[i]+8bp);

```

我们也可以把曲面离散化为数组, 然后如上对点进行着色.



```

import palette;
import three;
currentprojection=orthographic(8,5,9);
currentlight.background=black;
size(200);
triple[] a;
for(real i=-5;i<5;i=i+.25)
    for(real j=-5;j<5;j=j+.25)
        a.push((i,j,sin(i)*cos(j)));
for(int i=0; i < a.length; ++i)
    dot(a[i],Rainbow(a.length)[i]);

```

3.30 NURBS



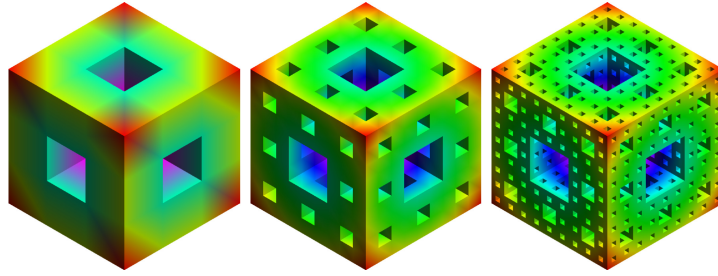
```
import three;
size(200);
currentprojection=perspective(615,-368,-233);
triple[] [] pts = {{
    (0.5, 0, -0.5),
    (0, 0, -0.5),
    (0, 1, -0.5),
    (0.5,1, -0.5),
    (1, 1, -0.5),
    (1, 0, -0.5),
    (0.5, 0, -0.5)
},{
    (0.5, 0, 0.7),
    (0, 0, 0.7),
    (0, 1, 0.7),
    (0.5, 1, 0.7),
    (1, 1, 0.7),
    (1, 0, 0.7),
    (0.5, 0, 0.7)
}, {
    (0.5, 0, 0.9),
    (0, 0, 0.9),
    (0, 1, 1.5),
    (0.5, 1, 1.5),
    (1, 1, 1.5),
    (1, 0, 0.9),
```

```

    (0.5, 0, 0.9)
  }, {
    (0.5, -0.1, 1),
    (0, -0.1, 1),
    (0, 0.5, 2),
    (0.5, 0.5, 2),
    (1, 0.5, 2),
    (1, -0.1, 1),
    (0.5, -0.1, 1)
  }, {
    (0.5, -0.3, 1),
    (0, -0.3, 1),
    (0, -0.3, 2),
    (0.5, -0.3, 2),
    (1, -0.3, 2),
    (1, -0.3, 1),
    (0.5, -0.3, 1)
  }, {
    (0.5, -1.5, 1),
    (0, -1.5, 1),
    (0, -1.5, 2),
    (0.5, -1.5, 2),
    (1, -1.5, 2),
    (1, -1.5, 1),
    (0.5, -1.5, 1)
  }
};
real[] uknot = {0, 0, 0, 1/4, 1/2, 3/4, 1, 1, 1};
real[] vknot = {0, 0, 0, 1/4, 1/2, 1/2, 3/4, 1, 1, 1};
real[][] weights={
  {1, .5, .5, 1, .5, .5, 1},
  {1, .5, .5, 1, .5, .5, 1},
  {1, .5, .5, 1, .5, .5, 1},
  {1, .5, .5, 1, .5, .5, 1},
  {1, .5, .5, 1, .5, .5, 1},
  {1, .5, .5, 1, .5, .5, 1}
};
draw(pts,uknot,vknot,weights,yellow);

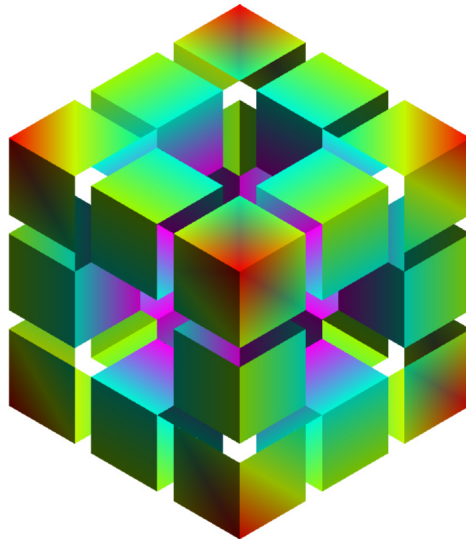
```

3.31 Sierpinski Sponge



Sierpinski Sponge(Sierpinski 垫) 与 Sierpinski Gasket(Sierpinski 海棉) 是 3D 分形几何的典型例子.

一阶的情形可以看成是魔方 $3 \times 3 \times 3 = 27$ 个小立方体挖掉中心的小立方体以及与这个中心小立方体相邻的 6 个小立方体以后所得到的立体图形. 因此, 最简单的做法是直接画出 20 个小立方体, 然后把它们移动到适当位置. 不过我们仔细观察图形, 可以发现实际上并不需要画出那些两个小立方体相接的面. 因此, 一阶的图有 48 个面是不用画出来的. 最终我们只需要 $120 - 48 = 72$ 个面.



类似的, 二阶的情形有可以看成 20 个一阶情形的立体图形去掉 48×8 个面, 然后移动到适当位置得到, 因此实际只要画出 $72 \times 20 - 48 \times 8 = 1056$ 个面. 同理, 三阶的图形只需画出 $1056 \times 20 - 48 \times 8^2 = 18048$ 个面. 根据这个递归关系我们很容易导出这种最少面数目的一个表达式 $2 \times 20^n + 4 \times 8^n, n = 0, 1, 2, \dots$

```
size(200);
import palette;
import three;
currentprojection=orthographic(1,1,1);
triple[] M={
    (-1,-1,-1),(0,-1,-1),(1,-1,-1),(1,0,-1),(1,1,-1),(0,1,-1),(-1,1,-1),(-1,0,-1),
    (-1,-1,0),(1,-1,0),(1,1,0),(-1,1,0),
    (-1,-1,1),(0,-1,1),(1,-1,1),(1,0,1),(1,1,1),(0,1,1),(-1,1,1),(-1,0,1)
```

```

};
surface[] Squares={
    surface((1,-1,-1)--(1,1,-1)--(1,1,1)--(1,-1,1)--cycle),
    surface((-1,-1,-1)--(-1,1,-1)--(-1,1,1)--(-1,-1,1)--cycle),
    surface((1,1,-1)--(-1,1,-1)--(-1,1,1)--(1,1,1)--cycle),
    surface((1,-1,-1)--(-1,-1,-1)--(-1,-1,1)--(1,-1,1)--cycle),
    surface((1,-1,1)--(1,1,1)--(-1,1,1)--(-1,-1,1)--cycle),
    surface((1,-1,-1)--(1,1,-1)--(-1,1,-1)--(-1,-1,-1)--cycle),
};
int[] [] index={
    {0,2,4},{0,1},{1,2,4},{2,3},{1,3,4},{0,1},{0,3,4},{2,3},
    {4,5},{4,5},{4,5},{4,5},
    {0,2,5},{0,1},{1,2,5},{2,3},{1,3,5},{0,1},{0,3,5},{2,3}
};
int[] Sponge0=array(n=6,value=1);
int[] eraseFaces(int n,int[] Sponge0){
    int[] temp=copy(Sponge0);
    for(int k : index[n]){
        temp[k]=0;
    }
    return temp;
}
int[] [] Sponge1;
for(int n=0;n<20;++n){
    Sponge1.push(eraseFaces(n,Sponge0));
}
surface s1;
int values;
real u=2/3;
for(int n1=0;n1<20;++n1){
    for(int k=0;k<6;++k){
        transform3 T=scale3(u)*shift(M[n1])*scale3(0.5);
        if(Sponge1[n1][k]>0 ){
            s1.append(T*Squares[k]);
            ++values;
        }
    }
}
}

```

```
s1.colors(palette(s1.map(abs),Rainbow()));
draw(s1);
write(values);
```

先看一阶情形的画法. 我们用含有 20 个点的数组 M 表现这 20 个点的相对位置关系. 我们用 6 元 0-1 数组表示六个面是否画出, 并且按照 $x, -x, y, -y, z, -z$ 这六个方向去看一个立方体的六个面是否画出, 比如我们看 $M[0]=(-1,-1,-1)$ 这个点, 它的 x, y, z 方向的面不用画出, 我们用 $index[0]=\{0,2,4\}$ 记录下这个情况. 接下去我们用 20×6 的二维数组表达这 20 个立方体中的每一个立方体的 6 个面是否画出, 举个例子, 对 $index[0]=\{0,2,4\}$, 我们把 6 元数组的第 0,2,4 个分量赋值为 0, 其它的是 1. 以后画面的时候, 如果是 0, 就不画出相应的面. 把这些抽象出来就是下面的代码.

```
int[] Sponge0=array(n=6,value=1);
int[] eraseFaces(int n,int[] Sponge0){
    int[] temp=copy(Sponge0);
    for(int k : index[n]){
        temp[k]=0;
    }
    return temp;
}
```

由于我们要对数组进行赋值, 因此要用 `copy()` 深复制出一个数组然后再赋值, 以便不会改动 `Sponge0`. 接下去的代码是把这些面依照移动到适当位置然后再 `append` 到 `surface s1` 中. 用这种 `append` 的方式以后, 就把这些正方形曲面的画出顺序交给 `asy` 去处理从而作到按照一定顺序画出以便作到消隐, 也可以使得我们可以对 `s1` 进行整体着色.

```
size(200);
import palette;
import three;
currentprojection=orthographic(1,1,1);
triple[] M={
    (-1,-1,-1),(0,-1,-1),(1,-1,-1),(1,0,-1),(1,1,-1),(0,1,-1),(-1,1,-1),(-1,0,-1),
    (-1,-1,0),(1,-1,0),(1,1,0),(-1,1,0),
    (-1,-1,1),(0,-1,1),(1,-1,1),(1,0,1),(1,1,1),(0,1,1),(-1,1,1),(-1,0,1)
};
surface[] Squares={
    surface((1,-1,-1)--(1,1,-1)--(1,1,1)--(1,-1,1)--cycle),
    surface((-1,-1,-1)--(-1,1,-1)--(-1,1,1)--(-1,-1,1)--cycle),
    surface((1,1,-1)--(-1,1,-1)--(-1,1,1)--(1,1,1)--cycle),
    surface((1,-1,-1)--(-1,-1,-1)--(-1,-1,1)--(1,-1,1)--cycle),
    surface((1,-1,1)--(1,1,1)--(-1,1,1)--(-1,-1,1)--cycle),
    surface((1,-1,-1)--(1,1,-1)--(-1,1,-1)--(-1,-1,-1)--cycle),
```



```

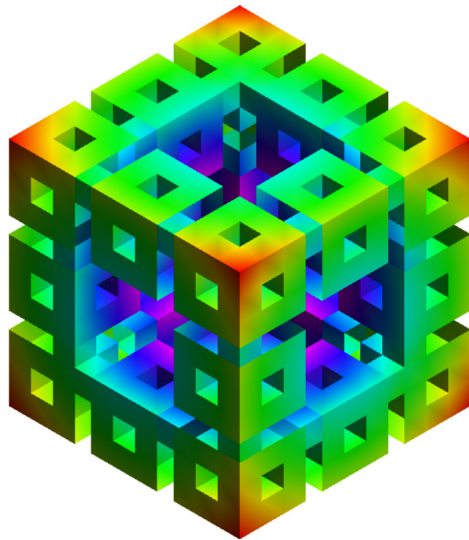
};
int[] [] SquaresPoints={
    {2,3,4,10,16,15,14,9},
    {0,7,6,11,18,19,12,8},
    {4,5,6,11,18,17,16,10},
    {2,1,0,8,12,13,14,9},
    {12,13,14,15,16,17,18,19},
    {0,1,2,3,4,5,6,7}
};
int[] [] index={
    {0,2,4},{0,1},{1,2,4},{2,3},{1,3,4},{0,1},{0,3,4},{2,3},
    {4,5},{4,5},{4,5},{4,5},
    {0,2,5},{0,1},{1,2,5},{2,3},{1,3,5},{0,1},{0,3,5},{2,3}
};
int[] Sponge0=array(n=6,value=1);
int[] eraseFaces(int n,int[] Sponge0){
    int[] temp=copy(Sponge0);
    for(int k : index[n]){
        temp[k]=0;
    }
    return temp;
}
int[] [] Sponge1;
for(int n=0;n<20;++n){
    Sponge1.push(eraseFaces(n,Sponge0));
}
int[] [] eraseFaces(int n,int[] [] Sponge1){
    int[] [] temp=copy(Sponge1);
    for(int k : index[n])
        for(int n1 : SquaresPoints[k])
            temp[n1][k]=0;
    return temp;
}
int[] [] [] Sponge2;
for(int n=0;n<20;++n){
    Sponge2.push(eraseFaces(n,Sponge1));
}
surface s2;

```

```

int values;
real u=2/3;
for(int n2=0;n2<20;++n2){
    surface s1;
    for(int n1=0;n1<20;++n1){
        for(int k=0;k<6;++k){
            transform3 T=scale3(u)*shift(M[n1])*scale3(0.5);
            if(Sponge2[n2][n1][k]>0 ){
                s1.append(T*Squares[k]);
                ++values;
            }
        }
    }
    transform3 T=scale3(u)*shift(M[n2])*scale3(0.5);
    s2.append(T*s1);
}
s2.colors(palette(s2.map(abs),Rainbow()));
draw(s2);
write(values);

```



二阶的情形本质上是把一阶情形得到的图形复制 20 个, 并且每一个按照 `index` 数组把一些面去掉, 然后再移动到适当位置. 这次根据 `index` 去掉的是一组面而不单单是一个面, 因此我们引进 `SquaresPoints`. 这个 `SquaresPoints` 记录了 6 个方向中分别包含的 `M` 中的哪些点. 于是这次的去掉面的代码如下所示.

```

int[] [] eraseFaces(int n,int[] [] Sponge1){

```

```

    int[] [] temp=copy(Sponge1);
    for(int k : index[n])
        for(int n1 : SquaresPoints[k])
            temp[n1][k]=0;
    return temp;
}

size(200);
import palette;
import three;
currentprojection=orthographic(1,1,1);
triple[] M={
    (-1,-1,-1),(0,-1,-1),(1,-1,-1),(1,0,-1), (1,1,-1),(0,1,-1),(-1,1,-1),(-1,0,-1),
    (-1,-1,0),(1,-1,0),(1,1,0),(-1,1,0),
    (-1,-1,1),(0,-1,1),(1,-1,1),(1,0,1), (1,1,1),(0,1,1),(-1,1,1),(-1,0,1)
};
surface[] Squares={
    surface((1,-1,-1)--(1,1,-1)--(1,1,1)--(1,-1,1)--cycle),
    surface((-1,-1,-1)--(-1,1,-1)--(-1,1,1)--(-1,-1,1)--cycle),
    surface((1,1,-1)--(-1,1,-1)--(-1,1,1)--(1,1,1)--cycle),
    surface((1,-1,-1)--(-1,-1,-1)--(-1,-1,1)--(1,-1,1)--cycle),
    surface((1,-1,1)--(1,1,1)--(-1,1,1)--(-1,-1,1)--cycle),
    surface((1,-1,-1)--(1,1,-1)--(-1,1,-1)--(-1,-1,-1)--cycle),
};
int[] [] SquaresPoints={
    {2,3,4,10,16,15,14,9},
    {0,7,6,11,18,19,12,8},
    {4,5,6,11,18,17,16,10},
    {2,1,0,8,12,13,14,9},
    {12,13,14,15,16,17,18,19},
    {0,1,2,3,4,5,6,7}
};
int[] [] index={
    {0,2,4},{0,1},{1,2,4},{2,3},{1,3,4},{0,1},{0,3,4},{2,3},
    {4,5},{4,5},{4,5},{4,5},
    {0,2,5},{0,1},{1,2,5},{2,3},{1,3,5},{0,1},{0,3,5},{2,3}
};
int[] Sponge0=array(n=6,value=1);

```

```

int[] eraseFaces(int n,int[] Sponge0){
    int[] temp=copy(Sponge0);
    for(int k : index[n]){
        temp[k]=0;
    }
    return temp;
}

int[] [] Sponge1;
for(int n=0;n<20;++n){
    Sponge1.push(eraseFaces(n,Sponge0));
}

int[] [] eraseFaces(int n,int[] [] Sponge1){
    int[] [] temp=copy(Sponge1);
    for(int k : index[n])
        for(int n1 : SquaresPoints[k])
            temp[n1][k]=0;
    return temp;
}

int[] [] [] Sponge2;
for(int n=0;n<20;++n){
    Sponge2.push(eraseFaces(n,Sponge1));
}

int[] [] [] eraseFaces(int n,int[] [] [] Sponge2){
    int[] [] [] temp=copy(Sponge2);
    for(int k : index[n])
        for(int n2: SquaresPoints[k])
            for(int n1: SquaresPoints[k])
                temp[n2][n1][k]=0;
    return temp;
}

int[] [] [] [] Sponge3;
for(int n=0;n<20;++n){
    Sponge3.push(eraseFaces(n,Sponge2));
}

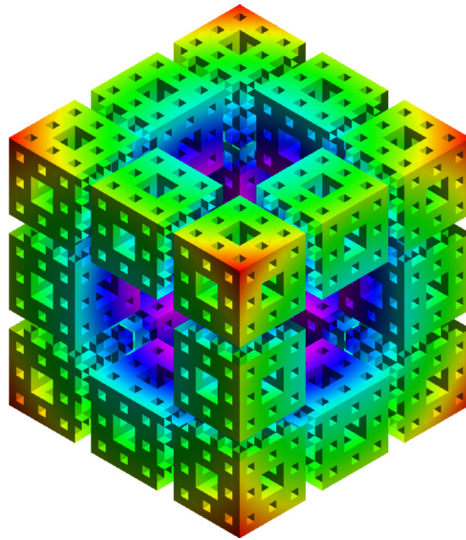
surface s3;
int values;
real u=2/3;
for(int n3=0;n3<20;++n3){

```

```

surface s2;
for(int n2=0;n2<20;++n2){
  surface s1;
  for(int n1=0;n1<20;++n1){
    for(int k=0;k<6;++k){
      transform3 T=scale3(u)*shift(M[n1])*scale3(0.5);
      if(Sponge3[n3][n2][n1][k]>0 ){
        s1.append(T*Squares[k]);
        ++values;
      }
    }
  }
  transform3 T=scale3(u)*shift(M[n2])*scale3(0.5);
  s2.append(T*s1);
}
transform3 T=scale3(u)*shift(M[n3])*scale3(0.5);
s3.append(T*s2);
}
s3.colors(palette(s3.map(abs),Rainbow()));
draw(s3);
write(values);

```



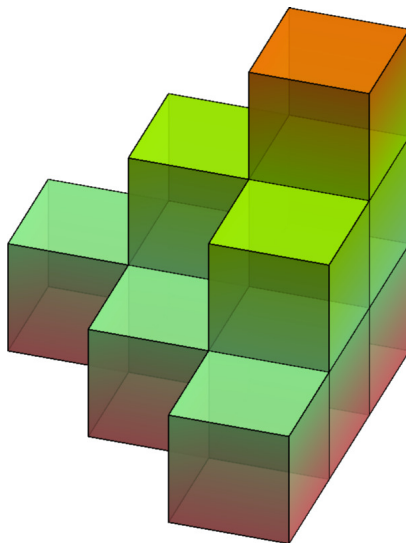
三阶情形也是建立在二阶情形上的, 原理是类似的.

这里顺便说说相关的历程. 2009 年 11 月 17 日, 笔者把 METAPOST 的 M3D 宏包中的 Sierpinski Sponge 和 Sierpinski Gasket 翻译成 asy 代码, 这个不难, 半天就完成, 提交到 Asymptote 在 SourceForge

的 Forum 以后, 立即就被采纳并且命名为 SierpinskiSponge.asy 和 SierpinskiGasket.asy, 不过接下去笔者发现 Sponge 的那个代码以及原来 M3D 里面关于 Sponge 的两个代码都不是最优的, 因此进行了一些改进, 改进后的结果本质上是把最优的一阶情形复制移动, 虽然不太完美, 不过这个改进还是被采纳了. 再接下去经过大半个月的努力, 终于成功实现了最少的画法, 不过可能代码本身还不是特别容易澄清, 因此上述画法的代码还没有被接纳. 这次写在 asy4cn 项目的这份文档里, 作为一个纪念, 因为笔者认为这是提交给 Asymptote 的 8 个例子³中最成功的一个.

3.32 一些示例

3.32.1 立方体堆成的三视图



```
size(200);
import three;
import palette;
currentprojection=orthographic(3,1,2);
int[] [] M={
    {1,2,3},
    {0,1,2},
    {0,0,1},
};
surface s;
for(int i=0;i<M.length;++i)
    for(int j=0;j<M[i].length;++j)
```

³这 8 个例子是 inlinemovie3.tex, torusanimation.asy, extrudedcontour.asy, NURBSsphere.asy, soccerball.asy, truncatedIcosahedron.asy, SierpinskiGasket.asy, SierpinskiSponge.asy——编者注

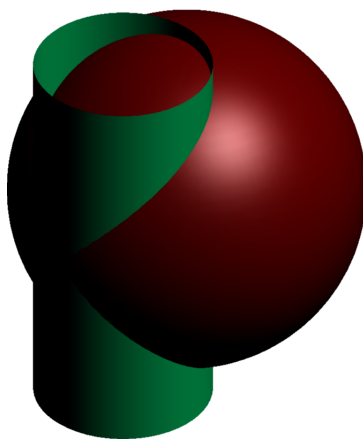
```

    for(int k=0;k<M[i][j];++k)
        s.append(shift(i,j,k)*unitcube);
s.colors(palette(s.map(zpart),Rainbow()+opacity(0.85)+yellow));
draw(s,meshpen=black+thick());

```

其中的二维数组 M 记录了在 3×3 个小方格的每一个堆放小立方体的数目.

3.32.2 Viviani 立体和 Viviani 曲线

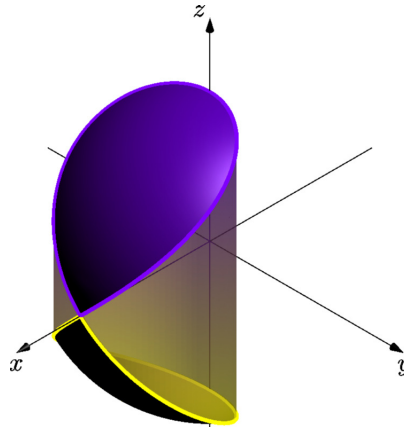


```

import solids;
currentprojection=orthographic(1,1,1);
size(200);
real a=4;
draw(scale3(a)*unitsphere,brown);
real h=2.2a;
draw(shift((a/2,0,-h/2))*scale(a/2,a/2,h)*unitcylinder,springgreen);

```

我们把球被圆柱截下的立体图形称为 Viviani 立体, 此处球的半径是 a , 圆柱的中心是 $(a/2, 0, 0)$, 半径是 $a/2$, 因此被圆柱截下的球面的这部分是定义在圆形区域上的曲面, 我们可以用第 129 页的化极坐标的方法来写出它的参数方程.



```

import palette;
import graph3;
currentprojection=orthographic(1,1,1);
size(200);
real a=4;
real S(pair t){
    return sqrt(a^2-t.x^2-t.y^2+realEpsilon);
}
triple V(pair t){
    real x=a/2+(a/2)*t.x*cos(t.y);
    real y=(a/2)*t.x*sin(t.y);
    real z=S((x,y));
    return (x,y,z);
}
triple v(real t){
    return V((1,t));
}
path3 p=graph(v,0,2pi);
path3 q=zscale3(-1)*p;
draw(p,purple+2pt);
draw(q,yellow+2pt);
surface sf=extrude(p,q);
sf.colors(palette(sf.map(zpart),Gradient(yellow,purple)+opacity(0.6)));
draw(sf);
surface s=surface(V,(0,0),(1,2pi),40,Spline);
draw(s,purple);
draw(zscale3(-1)*s,yellow);

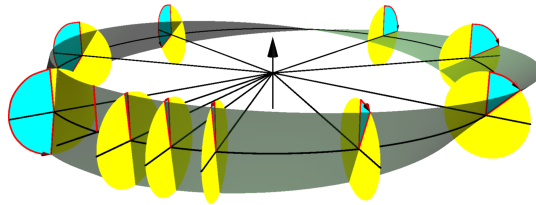
```



```
limits((-a-1)*(X+Y+Z),(a+2)*(X+Y+Z));
axes3("$x$","$y$","$z$",Arrow3());
```

`sqrt(a^2-t.x^2-t.y^2+realEpsilon)`; 是上半球球面的方程, 为了避免在后面调用 `sqrt` 函数时报错, 我们在 `sqrt` 表达式里面加上一个小小的正数 `realEpsilon`. 接下去由上半曲面的参数方程 $V(t)$ 立即得到上半曲面的边界 (称为 Viviani 曲线) 的方程, 利用反射得到下半曲面边界的曲线, 然后用 `extrude` 函数连接这两曲线就得到 Viviani 体的侧面.

3.32.3 Mobius 带原理的演示



```
import graph3;
size(200);
real R=6;
real r=1;
triple Mobius(pair w){
    real t=w.x;
    real l=w.y;
    return R*expi(pi/2,t)+l*expi(t/2,t);
}
surface s=surface(Mobius,(0,-r),(2pi,r),30,Spline);
draw(Circle(0,R,Z));
draw(s,lightgreen+opacity(0.6));
for(real c: new real[] {0,0.05pi,0.1pi,0.15pi,0.3pi,0.5pi,0.8pi,pi,1.4pi,1.6pi,1.9pi}){
    triple Center=R*expi(pi/2,c);
    triple U=r*expi(0,c);
    triple V=r*expi(c/2,c);
    triple A=Center+U;
    triple B=Center+V;
    draw(Center--A,red);
    draw(Center--B,red);
    draw(0--(R+r)*expi(pi/2,c));
    if(c!=0){
        draw(arc(Center,A,B),red,Arrow3(1mm));
    }
}
```

```

draw(surface(Center--A--arc(Center,A,B)--Center--cycle),cyan,nolight);
draw(surface(Center--A--arc(Center,A,B,Cw)--Center--cycle),yellow,nolight);
}
}
zaxis3(Arrow3());

```

其中的 $R \cdot \exp(i\pi/2, t)$ 其实是 $(R \cdot \cos(t), R \cdot \sin(t), 0)$, 它表示 Mobius 带的中心线, 这是一个 X-O-Y 平面的圆周. 然后一根长为 $2r$ 的小棍的中心绕着 Mobius 中心线转了 t 经度, 与此同时小棍的末端相对于小棍的中心余维度变为 $t/2$, 即小棍自旋的速度是小棍中心转动速度的一半. 因此我们用 $l \cdot \exp(i t/2, t)$; 表达这一事实.

第四章 一些论题

本章讨论一些 asy 不太常见的用法, 以方便有此需要的读者查阅.

4.1 把 asy 代码嵌入 tex 文件的编译方案

我们在第一章已经提到可以把 asy 代码放在 `\begin{asy}\end{asy}` 环境里面, 注意这里的 `\end{asy}` 一定要独占一行且要顶格写, 即之前不能有空格, 否则会报错. 至于编译方案则有两种.

4.1.1 常规方案

假定我们的 tex 文件是 main.tex, 那么依次用 (假定我们用 xelatex)

```
xelatex main.tex
asy main-*.asy
xelatex main.tex
```

4.1.2 用 latexmk 脚本的方案

我们的另外一种方案是用 latexmk.pl 这个 perl 脚本. <http://www.phys.psu.edu/~collins/software/latexmk/versions.html> 下载最新版本以后, 然后适当修改一下里面的内容. 如果我们专心只用 xelatex 编译, 那么把

```
$latex = 'latex %0 %S';
$pdflatex = 'pdflatex %0 %S';
```

改为

```
$latex = 'xelatex %0 %S';
$pdflatex = 'xelatex %0 %S';
```

就可以了, 然后再把这个 perl 脚本放在系统 PATH 变量的路径中, 或者用它替换掉旧版本的 latexmk.pl. (Linux 下的 TeXLive 2009 是在 `/usr/local/texlive/2009/texmf-dist/scripts/latexmk` 目录下). 然后把下面内容保存为 `~/.latexmkrc` 文件 (也可保存为 `latexmkrc` 直接放在当前目录下)

```
sub asy {return system("asy '$_[0]'");}
add_cus_dep("asy","eps",0,"asy");
add_cus_dep("asy","pdf",0,"asy");
add_cus_dep("asy","tex",0,"asy");
```

那么执行 (同样假定嵌入 asy 代码的 tex 文件名为 main.tex).

```
latexmk -pdf main.tex
```

那么就会用 xelatex 编译嵌入 asy 代码的 tex 文件, 如果 asy 环境里面的内容改变了, 生成的 main-*.asy 文件也就会相应的改变, latexmk.pl 脚本会追踪这个变化, 重新编译那些 main-*.asy 文件, 然后再用 xelatex 编译. 反之, 如果我们并没有改动 asy 环境里面的 asy 代码, 那么 latexmk 是不会调用 asy 去分别编译那些 main-*.asy 文件的, 从而节省了内存和编译的时间.

emacs 的用户可以定义一个简单的 lmk 函数, 然后通过 M-x lmk 来执行这个脚本.

```
(defun lmk() (interactive)
  (save-buffer)
  (compile (concat "latexmk -pdf " (file-name-nondirectory buffer-file-name)))
)
```

Windows 用户要保证装有最新的 perl, 建议用 strawberry-perl, 可以把 latexmk.pl 和 latexmk.bat 放在系统环境变量 PATH 所在路径中, .latexmkrc 文件可以放在 C:\Documents and Settings\ 您的用户名\Application Data 即

```
%USERPROFILE%\Application Data\
```

中 (不过似乎最好还是命名为 latexmkrc 放在当前目录), 然后执行

```
latexmk.bat -pdf main.tex
```

或

```
perl -S latexmk.pl -pdf main.tex
```

-S 选项是让 perl 在系统环境变量 PATH 里面找 latexmk.pl

4.2 动画

Asymptote 的动画分 gif 和 mpeg 以及 pdf 三种格式, 前面两种都是调用 ImageMagick 把若干张 eps 或 pdf 图片经过转换以后叠合起来, 而 pdf 格式的则是调用 LaTeX 的 animation 宏包去完成这个叠合与嵌入任务的.

我们先举 pdf 的例子. Asymptote 通过 animate 宏包调用 LaTeX 的 animation 宏包, 我们需要用 import animate, 然后定义一个数据类型为 animation 的 Ani.

```
import animate;
pair[] P={(5,0),(-8,50),(55,50),(100,0)};

defaultpen(fontsize(20pt)+1.5pt);
animation Ani;
int n=30;
for(int i=0;i<n;++i){
    picture pic;
    size(pic,400);
    real t=i/n;
    pair A=interp(P[0],P[1],t);
    pair B=interp(P[1],P[2],t);
    pair C=interp(P[2],P[3],t);
    pair U=interp(A,B,t);
    pair V=interp(B,C,t);
    pair W=interp(U,V,t);
    draw(pic,P[0]--P[1]--P[2]--P[3]);
    draw(pic,A--B--C,green);
    draw(pic,U--V,blue);
    draw(pic,P[0]..controls A and U.. W,red);
    Ani.add(pic);
}
Ani.pdf(delay=400,"control");
```

4.3 做 3D 动画

Asymptote 做 gif 动画是先生成一系列的 eps 或 pdf, 然后调用 ImageMagick 的 convert 功能把它们转成 gif 文件.

下面的例子改编自笔者提供的的 torusanimation.asy(见 examples 目录下的 animations 目录)

```
settings.prc=false;
settings.render=4;
import graph3;
import animation;
viewportmargin=(10,10);
currentprojection=orthographic((19,11,18));
currentlight=(0,5,5);
unitsize(1cm);
real R=3;
real a=1;
int n=4;
triple f(pair t) {
    return ((R+a*cos(t.y))*cos(t.x), (R+a*cos(t.y))*sin(t.x), a*sin(t.y));
}
surface s=surface(f, (0,0), (2pi,2pi), 8,8, Spline);
typedef triple function(real);
function g(int k){
    return new triple(real s){
        return ((R-a*cos(2*pi*s))*cos(2*pi/(1+k+s)),
            (R-a*cos(2*pi*s))*sin(2*pi/(1+k+s)),
            -a*sin(2*pi*s));
    };
}
animation A;
picture fig;
size3(fig,200);
for(int i=0; i < n; ++i){
    draw(fig,s,yellow);
    A.add(fig);
    for(int j=0; j <= i; ++j){
        draw(fig,graph(g(j),0,1,operator...),blue+linewidth(2));
    }
    A.add(fig);
}
```

```

}
A.movie(delay=400);

```

主要的改动是用高阶函数和匿名函数定义了与 i 相关的函数族 $g(i)$. 这些 $g(i)$ 生成的曲线合起来构成了环绕环形的曲线.

如果要做嵌入 pdf 文档里面的动画或独立的 pdf 动画, 则要调用 `animate` 宏包, 而这个 `animate` 宏包又是通过调用 LaTeX 的 `animate.sty` 宏包来完成这个任务的.

```

settings.prc=false;
settings.render=0;
settings.tex="xelatex";
import three;
import animate;
currentprojection=orthographic(1,1,1.3);
picture base;
triple[] [] P={
    {(-1,-1,-0.5), (-1,-0.3,0), (-1,0.3,0), (-1,1,-0.5)},
    {(-0.3,-1,0), (-0.3,-0.3,0.5), (-0.3,0.3,0.5), (-0.3,1,0)},
    {(0.3,-1,0), (0.3,-0.3,0.5), (0.3,0.3,0.5), (0.3,1,0)},
    {(1,-1,-0.5), (1,-0.3,0), (1,0.3,0), (1,1,-0.5)}
};
surface sf=surface(patch(P));
draw(base,sf,palecyan+opacity(0.8),nolight);
draw(base,sequence(new path3(int i){
    return sf.s[i].external();},sf.s.length), pink+1bp);
animation A;
real T=0.7;
int n=20;
for(int i=0;i<n;++i){
    picture pic;
    size(pic,250);
    add(pic,base);
    real t=interp(0.2,0.8,i/n);
    path3 u=sf.s[0].uequals(T);
    path3 v=sf.s[0].vequals(t);
    draw(pic,u,red);
    draw(pic,v,invisible);
    triple P=point(u,t);
    triple Q=point(v,T);
}

```

```

dot(pic,Q,green);
dot(pic,P,green);
triple dU=dir(u,t);
triple dV=dir(v,T);
draw(pic,surface(plane(.6dU,.6dV,P)),yellow+opacity(0.5),nolight);
draw(pic,P--P+.5dU,Arrow3());
draw(pic,P--P+.5dV,Arrow3());
draw(pic,P--P+cross(.5dV,.5dU),Arrow3());
A.add(pic);
}
label(A.pdf("controls,autoplay,loop",keep=true,delay=400));

```

也可以做基于 OpenGL 的动画, 下面例子源于笔者提供的 `inlinetex3.tex`(见 `examples` 目录下的 `animations` 目录) 到目前的 2.08 版本为止, Windows 的用户还需要用

```
asy -threads -framedelay=100
```

才能看到动画效果 (必要的时候调整一下 `framedelay` 的值, 默认是 0). 另外, 用 `r` 键可以让动画倒放, 用 `p` 键可以让动画正放.

```

settings.loop=true;
settings.autoplay=true;
import graph3;
import animate;
currentprojection=orthographic(1,-2,0.5);
animation A;
int n=25;
for(int i=0; i < n; ++i) {
    picture pic;
    size3(pic,6cm);
    real k=i/n*pi;
    real f(pair z) {return 4cos(abs(z)-k)*exp(-abs(z)/6);}
    draw(pic,surface(f,(-4pi,-4pi),(4pi,4pi),Spline),paleblue);
    draw(pic,shift(i*6Z/n)*unitsphere,yellow);
    A.add(pic);
}
A.glmovie();

```

上述例子中 `n` 控制帖子的数量, `k=i/n*pi` 这个量可以使得曲面 `f` 有一个波动的变化, `i*6Z/n` 使得球的高度变化. `A.add()` 函数把当前对应 `i` 状态的图形 `pic` 添加到 `animation A` 上面去.

4.4 做 3D 幻灯

用 asy 生成的已经嵌入 PRC 真 3D 的 pdf 文件不能直接用普通的图像嵌入命令 `\includegraphics` 嵌入到 `tex` 中. 这时可以把 asy 代码嵌入到 `tex` 文档生成真 3D. 虽然 Asymptote 调整了 `hyperref` 宏包的调用, 不过似乎仍然与 Beamer 有冲突, 有兴趣的读者可以测试下面的例子.

```
\documentclass[pdftex]{beamer}
\usetheme{Singapore}
\usepackage{asymptote}
\usepackage{graphicx}
\usepackage{CJKutf8}
\renewcommand{\figurename}{图}
\CJKtilde
\begin{document}
\begin{CJK}{UTF8}{Hei}
  \begin{frame}[fragile]
    \frametitle{嵌入 PRC}
    \begin{figure}
      \centering
      \begin{asy}[height=6cm]
size(200);
import three;
draw(unitsphere,red,render(compression=Low,merge=true));
\end{asy}
      \caption{请单击上图}
    \end{figure}
  \end{frame}
\end{CJK}
\end{document}
```

asy 提供了 `slide` 模式, 可以制作简单的幻灯, 虽然没有 beamer 华丽, 不过在对各种 TeX 引擎的支持, 在嵌入 PRC 真 3D, 嵌入 asy 画的图形, 嵌入动画, 嵌入多媒体等有极大的方便, 对那些喜欢简朴风格的用户, 用 asy 做幻灯不失为一种好的选择. 这时可以用 `asyinclude` 命令来嵌入其后的代码生成的真 3D.

```
tex preamble("
\usepackage{xeCJK}
%\setmainfont{DejaVu Sans}
\setmainfont{TeX Gyre Heros:style=Bold}
\setCJKmainfont{SimHei}
```

```
");
settings.tex="xelatex";
settings.user="stepping=true"; // 生成步进形式的幻灯

orientation=Landscape;
import slide;
import three; // 此处告诉 asy 调用 hyperref 和 movie15.
usersetting(); // 使得 settings.user="stepping=true"; 生效.
titlepen=fontsize(40)+brown; // 控制 title 的字体大小和颜色.
itempen=fontsize(20)+green; // 控制 item 的字体大小和颜色.
import x11colors; // Azure 颜色在 x11colors 里面定义了.
fill(background,box((-1,-1),(1,1)),Azure);
label(background,"",(0,startposition.y));

titlepage(title="PRC 真 3D 示例",
           author="F.A.Zhang",
           institution="数学系",
           date="\today",
           url="http://bbs.ctex.org");
title("Asymptote 开始支持真3D");
item("整个进展如下");
subitem("先是支持把 asy 生成的 prc 嵌入 pdf 中");
subitem("接着支持利用 OpenGL 不依赖于 Adobe Reader 预览");
item("希望有进一步的进展");
item("下面是一个 unitbox 的例子");
newslide();
asyinclude("unitbox",20cm);
```

asyinclude 中的 unitbox.asy 如下

```
// unitbox.asy
import three;
size(200);
defaultrender=render(compression=Zero,merge=true);
draw(unitbox);
dot((1,1,1),red);
arrow(Label("$A$ 点"),(1,1,1),-Z+Y);
```

如果想用 CJK 方式, 只需把开头设字体部分更改.

```
texpreamble("\usepackage{CJKutf8}
```

```

\AtBeginDocument{\begin{CJK}{UTF8}{hei}}
\AtEndDocument{\clearpage\end{CJK}}");
defaultpen(font("OT1","cmss","m","b")); // 这里可以改变英文字母字体
//defaultpen(Helvetica());
settings.tex="pdflatex";

```

下面列举一些 `asy` 的 `slide` 提供的函数, 更多的函数及其用法详见 `slide.asy` 以及 `examples` 目录下的 `intro.asy` 和 `slidedemo.asy`, `animations` 目录下的 `slidemovies.asy` 等文件.

`titlepage()` 首页.

`outline()` 摘要页.

`title()` 标题.

`item` 条目.

`subitem()` 子条目.

`equations(string s, pen p=itempen)` 数学公式.

`asyfigure()` 嵌入图形.

`asyinclude(string s, real xsize=0, real ysize=xsize)` 嵌入 `asy` 文件生成的图形.

`asywrite` 向 `asyfigure` 或 `asyinclude` 输入代码, 通常的用法是 `asyfigure(asywrite())` 或 `asyinclude(asywrite())`.

`asycode()` 抄录代码.

`skip()` 间隔.

`newslide()` 新起一页幻灯.

4.5 做动画幻灯

```

settings.tex="xelatex";
import slide;
import animate;
usersetting();
orientation=Landscape;
import x11colors;
fill(background,box((-1,-1),(1,1)),Azure);
label(background,"", (0,startposition.y));
viewportsize=pagewidth-2pagemargin;

```

```

newslide();
srand(seconds());
int[] A=sequence(1,50);
int[] arrange(int[] A){
    int n = A.length;
    for (int i = 0; i < n; ++i) {
        int j = i + floor(unitrand()*(n-i));
        int tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;
    }
    return A;
}

real epsilon=0.1;
real radius=0.4;
bool valid(pair[] arr, pair z){
    bool retval = true;
    for (pair z0 : arr) {
        retval = retval && (length(z0-z)>=2*radius+epsilon);
    }
    return retval;
}

picture[] RandomCircles(int[] A,real r){
    picture[] fig;
    for(int i=0;i<A.length;++i){
        picture pic;
        size(pic,0,500);
        int n =A[i];
        pair[] arr;
        for (int i = 0; i < n; ++i) {
            pair z;
            do {
                z =r*(unitrand()-1/2, unitrand()-1/2);
            } while (!valid(arr, z));
            arr.push(z);
            filldraw(pic,shift(z)*scale(radius)*unitcircle,red);
        }
        fig.push(pic);
    }
}

```

```

    }
    return fig;
}
import animate;
animation Ani;
for(int k=0;k<1;++k){
    A=arrange(A);
    for(int i=0;i<2*A.length;++i){
        if(i%2==0)
            Ani.add(RandomCircles(A,10)[floor(i/2)]);
        else{
            picture figure;
            size(figure,0,500);
            label(figure,format("${%d}$",A[floor(i/2)]),(0,0),gray+fontsize(400));
            Ani.add(figure);
        }
    }
    write(A);
}
label(Ani.pdf("controls,loop",delay=1200,multipage=false));

```

`controls` 表示我们采用控制键 (包括单步, 快进, 自动播放, 调整自动播放速度的键等等).

`autoplay` 表示我们让 pdf 默认自动播放.

`loop` 表示在自动播放时不断循环.

4.6 嵌入多媒体

```

//生成的 pdf 文件目前只能在 Windows 下用最新的 Adobe Reader 打开.
settings.tex="xelatex";
texpreable("\usepackage{xeCJK}
\setmainfont{Comic Sans MS}
\setCJKmainfont{SimHei}");
settings.user="stepping=true";
string[] foods={"cake","chicken"};
orientation=Landscape;
import slide;
import embed;

```

```

usersetting();
defaultpen(fontsize(100)+blue);
for(int i=0;i<foods.length;++i){
    newslide();
    label(graphic(foods[i]+".pdf"));
    display(embed.embed(foods[i]+".mp3",
        "poster,text="+"\bf{"+foods[i]+"}"));
}
titlepen=fontsize(60)+red;
titlepen=fontsize(40)+blue;
title("title");
item("item");
item("item");
subitem("subitem");

```

4.7 用 Asymptote 添加图片中的字母标签

应该说, Asymptote 的某些 3D 绘图功能还不够强大, 一些需要调用复杂函数的图形目前还不能由 Asymptote 绘制出来. 但我们可以用其他软件生成图片, 然后用 asy 添加中英文标签.

例如我们调用功能强大的 Mathematica 绘制出著名的极小曲面 Costa 曲面, 并且导出特定像素的 pdf 图像, 代码如下.

```

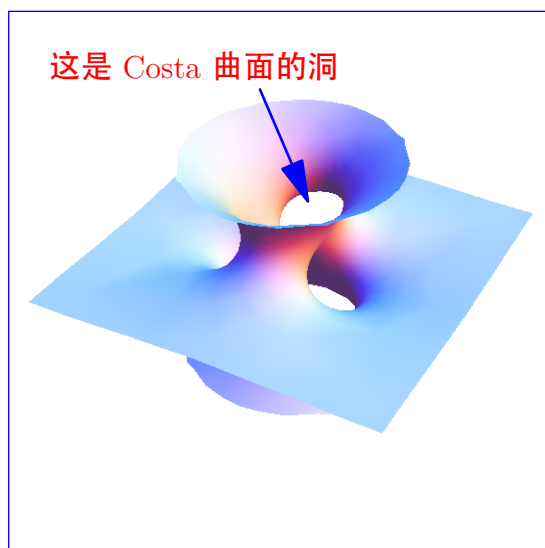
c = WeierstrassInvariants[{1/2, 0.5 I}][[1]];
P[z_] = WeierstrassP[z, {c, 0}];
Z[z_] = WeierstrassZeta[z, {c, 0}];
e = P[1/2];
Costa[u_, v_] := {
    (1/2)*Re[-Z[u + I*v] + Pi*u + Pi^2/(4 e) +
        Pi/(2 e)*(Z[u + I*v - 1/2] - Z[u + I*v - I/2])],
    (1/2)*Re[-I*Z[u + I*v] + Pi*v + Pi^2/(4 e) -
        Pi*I/(2 e)*(Z[u + I*v - 1/2] - Z[u + I*v - I/2])],
    (1/4)*Sqrt[2 Pi]*Log[Abs[(P[u + I*v] - e)/(P[u + I*v] + e)]]
};
sf = ParametricPlot3D[Costa[u, v], {u, 0, 1}, {v, 0, 1},
    Boxed->False, Axes->False, Mesh->False]
Export["Costa.pdf", sf, ImageResolution -> 200,
    ImageSize -> {200, 200}]
Import["Costa.pdf"]

```

那么我们可以用 Mathematica 右键提供的 Get Coordinates 获取图像点的坐标, 然后用 Asymptote 的

绘图功能添加一些新的元素.

```
// -*- coding:utf-8 -*-
settings.tex="xelatex";
texpreamble("\usepackage{xeCJK}
\setCJKmainfont{Adobe Heiti Std}
");
label(graphic("Costa.pdf"),position=(-2,-2),align=NE);
layer();
draw(Label("这是 Costa 曲面的洞",BeginPoint,red),(93,173)--(111,131),blue+1pt,Arrow());
shipout(bbox(0,blue));
```



其中 `layer()` 使得其后的添加的内容标在图片上层, 从而不会被图片遮挡. `align=NE` 放置嵌入的 `Costa.pdf` 文件, 会使得图片左下角的坐标是 $(0,0)$. 当然, 这里的嵌入与 Mathematica 捕获的坐标有些误差, 因此, 我们把图片往左下角移动了一点点, 即 `position=(-2,-2)`.

4.8 与 Mathematica 结合

Asymptote 由于还没有类似于 Mathematica 的 `RegionFunction` 命令, 因此对曲面裁剪还需要技巧, 不过由于 Mathematica 能输出 `obj` 这种 3D 格式, 而 `asy` 能调用 `obj` 宏包部分地读取 `obj` 文件里面的信息生成 3D 图形.

下面我们对 131 的例子用 Mathematica 与 `asy` 结合的方式重新生成.

首先用 Mathematica 很容易分别画出上下两个曲面, 且分别导出为 `sf1.obj` 和 `sf2.obj` 文件.

```
sf1 = Plot3D[8-x^2-y^2, {x, -2, 2}, {y, -2, 2},
  RegionFunction->Function[{x, y, z}, z >= x^2+3y^2],
```

```

    Mesh->False, Boxed->False, Axes->False, MaxRecursion->1,
    PlotPoints->60]
sf2 = Plot3D[x^2 + 3y^2, {x, -2, 2}, {y, -2, 2},
    RegionFunction->Function[{x, y, z}, z <= 8 - x^2 - y^2],
    Mesh->False, Boxed->False, Axes->False, MaxRecursion->1,
    PlotPoints->60]
Export["sf1.obj", sf1, "obj"]
Export["sf2.obj", sf2, "obj"]

```

然后如下调用 obj 宏包.

```

import obj;
import graph3;
size(0,200);
defaultrender=render(compression=Zero,merge=true);
currentprojection=perspective(6.8,5.6,10.6);
draw(obj("sf1.obj",yellow));
draw(obj("sf2.obj",blue));
limits((-2,-2,0),(2,2,9));
axes3("$x$","$y$","$z$",Arrow3());
shipout(bbox(0,blue));

```

